VŠB - Technická Univerzita Ostrava
Fakulta strojní
Katedra automatizační techniky a řízení

Tvorba webové aplikace s využitím HTML5

Building Web Application Using HTML5

Student: Bc. Martin Stříbný
Vedoucí diplomové práce: Ing. Pavel Smutný, Ph.D.

Ostrava 2012

VŠB - Technická univerzita Ostrava
Fakulta strojní
Katedra automatizační techniky a řízení

Zadání diplomové práce

Student: **Bc. Martin Stříbný**

Studijní program: N2301 Strojní inženýrství

Studijní obor: 3902T004 Automatické řízení a inženýrská informatika

Téma: Tvorba webové aplikace s využitím HTML5
Building Web Application Using HTML5

Zásady pro vypracování:

1. Seznamte se s dosavadním vývojem značkového jazyka HTML5 v kombinaci s kaskádovými styly verze 3 a skriptovacím jazykem JavaScript.
2. Seznamte se s nástrojem pro vývoj a nasazování vizualizačních a řídicích aplikací Control Web 6.1.
3. Navrhněte obecně změnu vizualizace dat v prostředí Control Web 6.1 s využitím jazyka HTML5.
4. Navrhněte úlohu demonstrující nové a pokročilé funkce jazyka HTML5 v prostředí Control Web 6.1.
5. Zhodnoťte dosažené výsledky a navrhněte směry dalšího řešení.

Seznam doporučené odborné literatury:

- The World Wide Web Consortium HTML5. [online]. [cit. 2011-03-15]. Dostupný z WWW: <http://www.w3.org/TR/html5/>
- The World Wide Web Consortium Cascading Style Sheets. [online]. [cit. 2011-03-15]. Dostupný z WWW: < http://www.w3.org/Style/CSS/>
- Pilgrim, M. Dive Into HTML5. [online]. [cit. 2011-03-15]. Dostupný z WWW: <http://diveintohtml5.org/>
- Piňo, T. Integrace HTML5 do systému pro správu obsahu Typo. Ostrava: VŠB-TUO. Diplomová práce. Dostupný z WWW: <http://hdl.handle.net/10084/78516>
- Smutný, P. Informační systémy v prostředí Internetu. Ostrava: Katedra automatizační techniky a řízení, VŠB-TU Ostrava, 2006. 94 stran. Doktorská práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavel Smutný, Ph.D.**

Datum zadání: 16.12.2011

Datum odevzdání: 21.05.2012

prof. Ing. Jiří Tůma, CSc.
vedoucí katedry

prof. Ing. Radim Farana, CSc.
děkan fakulty

Místopřísežné prohlášení studenta

Prohlašuji, že jsem celou bakalářskou práci včetně příloh vypracoval samostatně pod vedením vedoucího bakalářské práce a uvedl jsem všechny použité podklady a literaturu.

V Ostravě dne 21. května 2012

…….

podpis studenta

Prohlašuji, že

- jsem byl seznámen s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121\2000 Sb., autorský zákon zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo.
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen „VŠB-TUO“) má právo nevýdělečně ke své vnitřní potřebě diplomovou práci užít (§ 35 odst. 3.).
- souhlasím s tím, že diplomová práce bude v elektronické podobě uložena v Ústřední knihovně VŠB-TUO k nahlédnutí a jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že údaje o kvalifikační práci budou zveřejněny v informačním systému VŠB-TUO.
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít toto dílo v rozsahu § 12 odst. 4 autorského zákona.
- bylo sjednáno, že užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).
- beru na vědomí, že odevzdáním své diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., o vysokých a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, bez ohledu na výsledek její obhajoby.

V Ostravě: 21. května 2012

…….

podpis

Martin Stříbný

Bílá Bříza 458

Štěpánkovice, Svoboda 747 28

ANOTACE DIPLOMOVÉ PRÁCE

Hlavní část diplomové práce je zaměřena na vytvoření rozhraní mezi programem ControlWeb a webovým prohlížečem, tak aby bylo možné snadno změnit standardní statické webové rozhraní za rozhraní dynamické. Na straně ControlWebu je toho dosaženo přidáním speciálního kódu do běžící aplikace a na straně webového prohlížeče je nutné implementovat JavaScriptovou knihovnu, která se stará o přijetí dat a následné vykreslení měřidel.

Začátek práce je nejprve věnován popisu nových možností jazyka HTML5, kdy jsou některé z nich dále používány pro vizualizaci ve webovém prohlížeči. Zejména se jedná o element Canvas. Dále popisuji samotnou implementaci dynamického webového rozhraní v ControlWebu, a také použití webovém prohlížeči, a na závěr je popsaná jednoduchá testovací aplikace.

ANNOTATION THESIS

The main part of the thesis is focused on developing an interface between the program ControlWeb and web browser so that, you can easily change the standard static web interface for dynamic interface. On the ControlWeb is achieved by adding special code into a running application and the web browser must implement JavaScript library that provides for the adoption of data and subsequent rendering of gauges.

Beginning of my thesis is devoted to the description of the new possibilities of language HTML5, when some of them are also used for visualization in a web browser. In particular it deals with Canvas element. I describe the implementation of dynamic web interface in ControlWeb and also using of a web browse. The conclusion of thesis describes a simple test application.

Obsah

Seznam použitých značek a symbolů

AJAX	Asynchronous JavaScript and XML, označení pro technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich znovunačítání.
API	Application Programming Interface – Sbírka procedur, funkcí či tříd nějaké knihovny
CSS	Cascading Style Sheets – Kaskádové styly. Slouží pro definici vzhledu webové stránky.
CSS3	Nejnovější verze jazyka CSS
CW	ControlWeb
DOM	Document Object Model – objektový model dokumentu
DTD	Document Type Definition - Definice typu dokumentu, popis struktury XML
HTML	HyperText Markup Langure – Značkovací jazyk pro tvorbu webových stránek.
HTML5	Rozšiřující specifikace jazyka HTML.
JS	JavaScript – Programovací jazyk ve webových prohlížečích
JSON	JavaScript Object Station – Formát pro definici JavaScriptového objektu.
SQL	Structured Query Language - strukturovaný dotazovací jazyk
URL	Uniform Resource Locator
VML	Vector Markup Langure – Značkovací jazyk pro definici vektorových obrázků použitý v programu Internet Explorer.
XHTML	Verze HTML dokumentu kompatibilní s XML
XML	Extensible Markup Language

Úvod

Vizualizace procesů je v dnešní době automatizace důležitou součástí běhu většiny aplikací. Cílem vizualizace je umožnit lidem nahlédnout do procesů, které by běžně ani nepostřehli nebo je běžně nevnímají. Uživatelská rozhraní se běžně používají na lokálních počítačích, stále častěji však existuje poptávka po vizualizaci a vytvoření uživatelského rozhraní ve webovém prohlížeči, který je všude dostupný a jestliže je počítač zpracovávající data připojen k internetu, pak je možné sledovat aktuální stav odkudkoli na světě.

Webové uživatelské rozhraní dokáže generovat také program ControlWeb, který jej vytváří pomocí obrázků, které jsou odesílány do webového prohlížeče. Takové rozhraní je poměrně snadné na vytvoření avšak má několik nevýhod. Přenesení obrázku po síti je datově náročné, generování obrázku může být při běhu realtime aplikace náročné i časově, webová stránka se musí pravidelně obnovovat a narušuje tak uživatelský komfort, frekvence obnovování nemůže být příliš velká (méně než sekundu) a našlo by se toho více. Řešením tohoto problému je nepřenášet obrázky ale přenášet pouze data. Dlouhou dobu nebylo možné ve webové stránce generovat grafické objekty, ale to se změnilo z příchodem HTML5.

HTML5 není pouze návrh nového značkovacího jazyka, ale obsahuje celou sadu nových prvků, které lze na webu používat. HTML jako takové je pouze jednoduchý značkovací jazyk, kde každá značka má svůj sémantický význam, který mohou využít jak uživatelé tak např. vyhledávače nebo různé čtečky a další aplikace. HTML5 nepřináší pouze nové značky ale hlavně také nové funkce, které budou webové prohlížeče implementovat, a které umožní vytvářet mnohem sofistikovanější webové aplikace než doposud.

Na začátku práce se nachází seznámení se samotným HTML, základní syntaxí a pohledem na HTML jako na značkovací jazyk. Následně je popsáno několik rozdílů mezi HTML5 a předchozími verzemi tohoto jazyka. Jsou zde popsány nové značky a jejich sémantický význam. Důkladně jsou zejména značky audio, video a canvas, které by mohli po rozšíření nahradit většinu aplikací, kde je nyní nutné použít externí doplňky. Současně jsou zde předvedeny ukázky využití nových prvků.

Hlavní část práce se pak věnuje vytvořenému rozhraní mezi ControlWebem a webovým prohlížečem. Je zde detailně popsáno jakým způsobem probíhá komunikace a jak je sestavena odpověď serveru na požadavek o nová data. Samotné rozhraní vytváří pouze jakousi mezivrstvu kdy běžný tvůrce aplikací v ControlWebu nemusí vůbec znát, jakým způsobem funguje, ale pouze jakým způsobem ji má použít. Dále je zde popisováno jakým způsobem je možné vytvořit nová měřidla a celou aplikaci tak velice jednoduše rozšířit o nové vizualizační prvky.

Ke konci práce je pak představena implementace nového webového rozhraní na jednoduchém příkladu, kdy je nutné pouze zkopírovat připravené kódy do ControlWebu a určit co se má překreslit novými prvky. O nahrazení starého rozhraní novým se pak již postará samotná aplikace.

1 Vývoj HTML5

V této kapitole se nejdříve popíšu samotný jazyk HTML, na který dále navážu. Dále budou popsány rozdíly mezi současným standardem HTML5 a předchozími verzemi. Některé nové značky jako je audio, video nebo canvas budou důkladně popsány a ukázány způsoby jak s nimi pracovat.

1.1 Co je HTML?

HTML je značkovací jazyk založený na SGML. Jednotlivé značky jazyka dávají svému obsahu určitý sémantický význam. Některé značky jsou však použity jen pro formátování dokumentu. V HTML existuje celá řada značek, které mají svůj význam nebudu je tedy všechny uvádět. Níže je uveden příklad jednoduchého HTML dokumentu.

```
<html>
<head>
<title>Titulek stránky</title>
</head>
<body>
<h1>Nadpis první úrovně</h1>
<p>Odstavec</p>
<a href="url">odkaz</a>
</body>
</html>
```

V tomto příkladu je uvedeno několik značek, které mají definovaný význam. Probereme si je jeden po druhém.

- html – Určuje že se jedná o html dokument.
- head – Určuje, že cokoli uvedené v tomto bloku se týká hlaviček dokumentu, které nebudou zobrazeny.
- title – Definuje titulek stránky, který je uveden na liště v prohlížeči.
- body – Vše v tomto elementu je tělo dokumentu, které se bude nějak zobrazovat.
- h1 – Text zapsaný v této značce je nadpis první úrovně.
- p – Určuje, že vše v této značce patří do tohoto odstavce.
- a – Definuje odkaz, který povede na adresu uvedenou v atributu href a text zapsaný v tomto elementu bude tvořit tělo odkazu, na které půjde kliknout.

V HTML jsou značky zapsány v ostrých závorkách <> a existují značky, které jsou párové nebo nepárové. Všechny párové značky musí být uzavřeny. Například odstavec začíná značkou a je uzavřen značkou . Uzavírací značka je shodná s otevírací jen je v ní navíc obsaženo lomítko před názvem značky. Typickou nepárovou značku je například <br>, která způsobí zalomení řádku. Pro tuto značku neexistuje uzavírací varianta a v dokumentu stojí vždy jen sama o sobě.

K jednotlivým elementům je také možné uvádět atributy. Ty se zapisují vždy do otevírací značky a to následujícím způsobem:

```
<znacka atribut="hodnotaAtributu">obsah znacky</znacka>
```

V předcházejícím příkladě byla pomoci atributu href definována adresa odkazu. Jednotlivé atributy mohou mít různé významy, ale jejich popis není předmětem této práce. Hodnotu atributu je vhodné psát do uvozovek. Některé specifikace jazyka HTML toto přímo nevyžadují a webové prohlížeče umí pracovat i s variantami bez uvozovek. V takovém případě může hodnota atributu obsahovat jen alfanumerické znaky.

1.2 Rozdíly HTML5 oproti předchozím verzím

HTML5 je možné s předchozími verzemi srovnat s několika pohledů. Z hlediska délky kódu, definice dokumentu, způsoby zápisu značek, ze sémantického hlediska tak i z funkčního. Jednotlivé pohledy budou popsány níže.

Definice typu dokumentu a kódování

HTML5 nabízí snadnější definice typu dokumentu bez složitého popisování dtd. Dále se také zjednodušila definice kódování webové stránky, kdy se vždy musel uvést typ obsahu a dále kódování. Níže je na ukázkách vidět rozdíl mezi jednotlivými verzemi HTML.

Ukázky jednoduchých webů v různých verzích

HTML 4.01 strict

```
<!doctype html public "-//W3C//DTD HTML 4.01//EN">
<html lang="cs">
  <head>
```

```
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
    <title>Titulek</title>
  </head>
  <body>
    <p>Cokoliv.
  </body>
</html>
```

XHTML 1.0 strict

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="cs">
  <head>
    <title>Titulek</title>
  </head>
  <body>
    <p>Cokoliv.</p>
  </body>
</html>
```

HTML5

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Example document</title>
  </head>
  <body>
    <p>Example paragraph</p>
  </body>
</html>
```

XHTML5

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Example document</title>
```

```
  </head>
  <body>
    <p>Example paragraph</p>
  </body>
</html>
```

Z výše uvedených ukázek je vidět vývoj značkovacího jazyka HTML. Ve verzi 4.01 není třeba uzavírat nepárové tagy a dokonce ani některé párové tagy, jejichž konec vyplývá z kontextu. Ve verzi XHTML 1.0 byla snaha upravit formát HTML, tak aby byl kompatibilní se standardem XML. Všechny tagy ať už párové nebo nepárové museli být ukončeny lomítkem.

Ve verzi HTML5 se nepárové tagy opět nemusí uzavírat a zjednodušila se definice HTML dokumentu. Aby bylo možné HTML5 zpracovávat také jako XML, tak byla vymyšlená verze XHTML5, která musí splňovat standard XML, takže na počátku dokumentu musí být uvedena XML hlavička a všechny tagy musí být stejně jako ve verzi XML 1.0 zakončeny. Párové tagy ukončovací značkou a nepárové tagy lomítkem v úvodní značce na konci.

Nové značky

V HTML5 přibyla také spousta nových elementů, které určují jak různé sémantické významy, které v HTML4 chyběly tak přibyli také elementy, které mají také své reálné funkce, se kterými se dá pracovat. Všechny nové elementy jsou přidány na základě praxe a možnosti skutečného využití na webových stránkách.

Všechny jsou k nalezení na http://www.w3.org/TR/2009/WD-html5-diff-20090423/. Zde uvedu pouze seznam značek, které mají nejčastější využití.

- section – Představuje obecnou část dokumentu.
- article – Představuje článek dokumentu. (např. na blogu)
- aside – Část dokumentu, která nesouvisí s hlavním obsahem. (Text v bočním panelu)
- header – Záhlaví webové stránky.
- footer – Zápatí webové stránky.
- nav – Představuje navigaci na webové stránce.
- dialog – Slouží k zapsání rozhovoru.
- figure – Značka spojuje mediální obsah a jeho popis.

- video – Vloží do webové stránky přehrávač videa.
- audio – Vloží do webové stránky přehrávač zvuku.
- canvas – Představuje jakoukoli grafiku, která se vytváří pomocí JS.

Dříve se pro formátování celé stránky používal nejčastěji element div s určitým parametrem id nebo class. Byly tak často k vidění například zápisy ve tvaru:

```
<div id="header">...</div>
<div id="article">...</div>
<div id="nav">...</div>
<div id="footer">...</div>
```

Tyto zápisy nemají žádný sémantický význam a nelze je tak nijak strojově zpracovat. Nově lze díky HTML5 psát tyto často používané konstrukce jako:

```
<header>...</header>
<article>...</article>
<nav>...</nav>
<footer>...</footer>
```

Hlavní přínos je v tom, že tyto elementy již mají sémantický význam. Pro běžného uživatele to nemá v podstatě žádný význam. Vše se zobrazí stejně jako v případě použití divů. Pro kodéra to také není nijak velká změna, jediný rozdíl je v tom jak elementy zapíše a jak se na ně bude odkazovat např. v CSS nebo JS. Ve druhém případě ušetří pár znaků, ale dále se s dokumentem pracuje prakticky stejně. Hlavní přínos tohoto zápisu je však v tom, že jej lze mnohem snadněji strojově zpracovat. Z toho mohou těžit například webové vyhledávače, různé asociativní technologie, nebo jiné služby, které nějakým způsobem zpracovávají jiné webové stránky. A tvůrce webových aplikací by měl myslet jak na každého uživatele, tak na webové vyhledávače.

Zrušené elementy

Následující elementy byly zrušeny, protože k dosažení stejné funkčnosti je lepší použít CSS.

- basefont – určuje základní font pro celou HTML stránku
- big – zvětší text
- center – vycentruje text
- font – určí formátování textu

- s – přeškrtne text
- strike – také přeškrtne text
- tt – vypíše text strojopisem
- u – podtrhne text

Tyto prvky nejsou v HTML5, protože jejich použitím je negativně ovlivněna přístupnost pro koncového uživatele.

- frame – Zastupuje jeden rám dokumentu.
- frameset – Zastupuje skupinu rámů - slouží pro formátování rámů na stránce.
- noframes – Obsah zobrazený v případě že rámy nejsou v prohlížeči podporovány.

Následující elementy byly zrušeny, protože jejich použitím vznikal zmatek nebo je možné nahradit je jinými elementy.

- acronym – Bylo zrušeno z důvodů nejasností kdy jej použít. Nově je možné použít pouze abbr.
- applet – Zastaralý tag nahrazen novějším object.
- isindex - Může být nahrazeno použitím ovládacích prvků formuláře.
- dir – Bylo nahrazeno novějším ul.

Vylepšené formuláře

V současné době není možné standardním způsobem definovat, jaký obsah má být ve formulářovém políčku input zadán. Existují v zásadě jen dva typy obsahu a to buď text, nebo heslo, kde je možné nanejvýš omezit maximální délku vepsaného textu. Pokud chcete vkládanou hodnotu nějak jinak omezit nebo přesněji definovat musíte sáhnout již po JavaScriptu. V HTML5 se hodnoty atributu type značně rozšiřují a umožňují definovat snad vše co je potřeba. Níže je seznam hodnot atributu type.

- datetime – datum a čas
- datetime-local – datum a čas na lokálním počítači
- date – datum
- month – měsíc
- week – týden
- time – čas
- number - číslo
- range – zobrazí táhlo umožňující zadat rozsah číselných hodnot, které se nastaví parametry min a max

- email – pole pro zadání e-mailové adresy (s ověřením, zda je formát správný)
- url – URL adresa
- search – vyhledávací políčko
- tel – slouží pro zadání telefonního čísla
- color – pole s výběrem barvy a převedením do jejího textového formátu

Do HTML dokumentu je tak možné psát například

```
<input type="number" name="cena">
```

a webový prohlížeč se už postará o to, aby do pole bylo vloženo skutečně jen číslo a né jiná hodnota.

Mezi další atributy patří:

- pattern – povolený obsah textového políčka (určuje regulární výraz, kterému musí odpovídat obsah políčka),
- autocomplete – automatické doplňování pole dle již dříve zadaných hodnot (logická hodnota),
- min, max – rozsah, ve kterém se může číselná hodnota pole pohybovat,
- require – atribut určuje, zda musí být políčko vyplněno,
- placeholder – text, který bude v políčku, dokud se na něj neklikne.

Další změny

Změnami si neprošli pouze elementy ale také některé atributy elementů, jejichž změny jsou popsány na výše uvedené adrese. Výše jsou popsány pouze nejdůležitější změny oproti předchozím verzím.

Více změn naleznete v [HTML 5 differences from HTML 4, 2009]

1.3 Příklady některých nových elementů

V HTML 5 je definována spousta nových objektů. Níže je seznam těch, které v současné době asi nejvíc chybí.

Element figure

Element figure slouží ke svázání jakéhokoli multimediálního souboru s libovolnou částí webové stránky. Popis multimediálního objektu, tak nemusí být vázán pouze na

atributy, jako jsou alt nebo title ale je možné je určit přímo z textu na webové stránce. Příklad použití:

```
<figure>
 <video src="ogg"></video>
 <legend>Popis videa</legend>
</figure>
```

Tento zápis určuje, že element video popisuje text uvedený v elementu legend.

Element dialog

Na zpravodajských webech se velice často stává, že je citován rozhovor redaktora s hostem. Pro tyto případy byla vymyšlená nová značka dialog, jejíž použití je vidět níže.

```
<dialog>
 <dt> Mluví tazetel
 <dd> Odpovídá tázaný
 ...
 <dt> Mluví tazetel
 <dd> Odpovídá tázaný
</dialog>
```

Pomocí značky dialog je možné snadno zjistit, že se jedná o rozhovor a určit kdo se koho ptá. Díky tomu je takovýto text možné také mnohem lépe strojově zpracovat než kdyby byl zapsán v obecném elementu div a strukturován jen pomocí odstavců.

Video element

V počátku HTML kdy přenosová rychlost byla v řádech kilobajtů, nemohl nikdo počítat s masivním rozšířením videa na webu. Dnes je však situace trochu jiná a video je na každém webu. Dosud bylo možné vložit video pomocí elementu object nebo embed a pro jeho vykreslení byl použit nějaký doplněk ve webovém prohlížeči. Nejčastěji se však setkáme s variantou kdy je použit nějaký přehrávač naprogramovaný ve flashi, který je vložený do HTML dokumentu jako obecný objekt, pro který musí být nainstalován patřičný plugin ve webovém prohlížeči. Výhodou flashe je, že je nainstalován téměř v každém webovém prohlížeči vyjma některých firemních počítačů.

Zápis do HTML, který funguje ve všech prohlížečích, vypadá zhruba takto:

```
<object width="480" height="385">
    <param name="movie"
value="http://www.youtube.com/v/siOHh0uzcuY&hl=en_US&fs=1&"></param>
    <param name="allowFullScreen" value="true"></param>
    <param name="allowscriptaccess" value="always"></param>
    <embed src="http://www.youtube.com/v/siOHh0uzcuY&hl=en_US&fs=1&"
    type="application/x-shockwave-flash" allowscriptaccess="always"
    allowfullscreen="true" width="480" height="385"></embed>
</object>
```

Tento zápis působí jako šifra, většina kodérů se naučila tento kód používat metodou „copy & paste“ s následnou úpravou patřičných parametrů. V HTML5 je celý tento zápis zjednodušen do jednoho jediného řádku a výsledná funkčnost je prakticky stejná.

```
<video src="video.ogg" controls width="480" height="385"></video>
```

Při použití tagu video není třeba navíc žádný plugin, protože o vykreslení videa se stará samotný prohlížeč. V daném zápise je určena šířka a výška vide a pomocí atributu controls řekneme prohlížeči aby použil své výchozí ovládací prvky.

Tento jednoduchý zápis je však zatím pouze teorie, které se snad někdy v budoucnu dočkáme. Za současných okolností se tvůrci webových prohlížečů nemohou dohodnout, jaký formát pro přehrávání videa se bude na webu používat. Momentálně se diskutuje nad třemi formáty, které jsou:

1. **MPEG 4 (.mp4, .m4v)** – pracuje s video kodekem „H.264“ a s kodekem „AAC“ pro audio.
2. **OGG (.ogv)** – pracuje s kodeky „Theora“ pro video a „Vorbis“ pro audio.
3. **WebM (.webm)** – pracuje s kodekem „VP8“

Každý webový prohlížeč pokládá jiný formát za ten nejlepší pro použití na internetu. Každý z těchto formátů má své výhody i nevýhody, kterými se nebudu v této práci zabývat, protože by bylo třeba prozkoumat jednotlivé formáty do hlouby a porovnat je z mnoha hledisek od srovnání kvality po srovnání ceny a složitosti při implementaci.

Přehled podpory jednotlivých formátů

- **OGG (Theora+Vorbis)** – Firefox 3.5+, Opera 10.5+, Chrome 5.0+
- **MP4 (H.264+ACC)** – Safari 3.0+, Chrome 5.0+, iPhone 3.0+, Android 2.0+

- **WebM:** Chrome 6.0+

Oznámená podpora v prohlížečích:

- **MP4 (H.264+ACC)** – IE9
- **WebM**: IE9 (pokud si uživatel nainstaluje kodek VP8), Firefox 4.0+, Opera 11.0+, Android (přislíbeno, že v některé z budoucích verzí podpora bude)

V praxi se tak mnohem častěji setkáte se zápisem jako:

```
<video controls width="480" height="385">
        <source src="video.webm" type="video/webm">
        <source src="video.ogg" type="video/ogg">
        <source src="video.mp4" type="video/mp4">
</video>
```

Kde je možné zadat více zdrojů s různými formáty videa. Pro tvůrce webu to sice není nepřekonatelná překážka avšak značně obtěžující. Jedno video musí být zakódováno do třech různých formátů, aby mohlo být bezpečně zobrazeno na všech zařízeních. Do budoucna je však pravděpodobné, že se video formát sjednotí pro všechny webové prohlížeče. Nejlépe je na tom snad nový formát WebM, který vlastní google a je licencován jako open-source a veškeré jeho patenty jako royalty-free, je tudíž zdarma k dispozici pro kohokoli. Navíc většina prohlížečů již přislíbila jeho podporu.

Tag video obsahuje tyto parametry:

- src – cesta k videu. (je možné určit i elementem source),
- width – šířka videa,
- height – výška videa,
- controls – logický atribut určující zda se mají zobrazit výchozí ovládací prvky (je možné si vytvořit vlastní ovládací prvky jako běžné elementy a video ovládat pomocí JavaScriptu),
- poster – url adresu obrázku, který má být zobrazen před spuštěním videa,
- autoplay – logický atribut určující zda se má video spustit okamžitě,
- autobuffer – logický atribut určující zda se má video stahovat ihned po stažení, stránky (pokud není uveden video se začne stahovat až po jeho spuštění),
- look – logický atribut určující zda se má video přehrávat ve smyčce.

Samotný element video má vzhled ovládacího rozhraní definován prohlížečem a pomocí jednotlivých atributů je možné určit základní chování videa. Pro běžné přehrávání

je toto nastavení naprosto dostačující, pokud si však chcete zvolit vlastní vzhled ovládacích tlačítek nebo z jakýchkoli důvodů chcete přímo ovládat přehrávané video, pak musíte sáhnout po JavaScriptu, který toto umožňuje.

Vlastní ovládání videa

Vlastní ovládací tlačítka je možné vytvořit z jakéhokoli HTML elementu. Nejlépe jsou pro to ze sémantického hlediska vhodné elementy button, které se s pomocí CSS nastylují do podoby jaká vám bude vyhovovat. Samotné ovládání videa je pak nutné řešit pomocí JavaScriptu, který k tomu poskytuje spoustu metod. Nejdůležitější parametry a metody uvádím níže. Jednotlivé metody a parametry jsou definované přímo na elementu video, který je možné získat například takto:

```
var prahravac = document.getElementsByTagName("video")[0];
prehravac.play();
```

Tento kód vrátí první element video, který se nachází na stránce. Samozřejmě je možné jednotlivým elementům přiřadit Id a vyhledávat jej pomocí něj nebo využít jakýkoli jiný způsob vyhledávání elementů v HTML dokumetu.

Nejdůležitější metody objeku video:

- play() – zajistí spuštění videa,
- stop() – zajistí zastavení videa,
- load() – začne video stahovat ze serveru,
- currentTime – čas v sekundách určující aktuální místo na časové ose videa (nastavením hodnoty dojde k přetočení vide),
- duration – celkový délka videa v sekundách (pouze pro čtení, pokud data nebyla načtena, pak vrací 0, pokud je přenos streamován vrací Inf),
- volume – nastavuje hlasitost v rozmezí 0.0 – 1.0,
- muted – logická hodnota umožňující vypnout zvuk nezávisle na aktuální hlasitosti.

Pomocí výše zmíněných metod je možné video libovolně ovládat dle akcí uživatele nebo jakékoli JavaScriptové aplikace. Více o elementu video v [Sládek Jan, 2010]

Element Audio

S elementem audio je to v zásadě stejné jako s elementem video. Dnes se pro přehrávání videa nejčastěji používá opět flash, ve kterém je naprogramován přehrávač a do HTML dokumetu je vložen stejnou konstrukcí jako video. Pro vložení do stránky se používá zápis:

```
<audio src="hudba.mp3" controls></audio>
```

Stejně jako u elementu video není to ani u audia tak jednoduché. Opět jsou k dispozici 3 formáty (Ogg Vorbis, MP3, WAV), a u žádného není shoda napříč všemi prohlížeči

- **Firefox 3.5+** – Ogg Vorbis, WAV
- **Safari 4+** – MP3, WAV
- **Chrome 3+** – Ogg Vorbis, MP3
- **Opera 10.5+** – Ogg Vorbis, WAV

Stejně jako v případě videa je možné připojit více zdrojů pro přehrávání pomocí elementu source. Pro přehrání audia je nutné mít nejméně 2 zdrojové soubory.

```
<audio>
        <source type="audio/ogg" src="soubor1.ogg" />
        <source type="audio/mpeg" src="soubor1.mp3" />
</audio>
```

Další atributy elementu audio jsou:

- preload – logický atribut určující zda se má video načítat předem (analogie s autobuffer u videa)
- autoplay – logický atribut určující zda se má zvuk ihned spustit
- loop – logický atribut určující, zda se bude zvuk přehrávat ve smyčce
- controls – logický atribut určující, zda se mají použít výchozí ovládací prvky prohlížeče

Stejně jako u elementu video je možné i u audia ovládat zvuk pomocí JavaScriptu. Slouží k tomu stejné metody jako u videa, proto je zde nebudu opět rozvádět.

1.4 Element Canvas

Značka canvas, jak už z názvu vypovídá je plátno, na které je možné něco vykreslit, jeho použití je navrženo pro vykreslení libovolné 2D grafiky. Vykreslení se provádí pomocí JavaScriptu a je možné jej využít například pro vykreslení obrázků, tvorbu grafů, animací, interaktivních prostředí, her apod. Značka sama o sobě nemá žádnou funkčnost, ta se musí až následně doprogramovat pomocí JavaScriptu.

Jednoduchý příklad jak se element canvas používá je ukázán níže:

```
<canvas id="idCanvasu" width="300" height="150">
Obsah, pro prohlížeče, které canvas nepodporují.
</canvas>
```

Tímto kódem jsme vložili do HTML stránky plátno o velikosti 300 px*150 px, do kterého můžeme dále kreslit. Do elementu canvas je možné kreslit jak 2d tak 3d grafiku, záleží jen na kontextu, o který si řekneme. V dalším textu se budu zabývat pouze kreslením 2d grafiky.

```
// získáme element canvasu
var elem = document.getElementById('myCanvas');

// ověříme jestli element existuje a podporuje vlastnost getContext
if (elem && elem.getContext) {
   // získáme 2d context
   var context = elem.getContext('2d');
    // A nakreslíme obdélník
    // Zadáme souřadnice x,y následované šířkou a výškou.
   context.fillRect(0, 0, 150, 100);
}
```

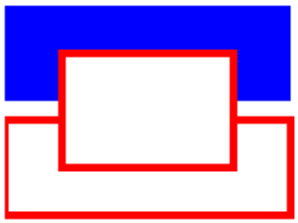
Tímto kódem nakreslíme do plátna obdélník na souřadnicích x = 0, y = 0, šířce 150 px a výšce 100 px. Obdélník bude vybarven černou barvou, což je výchozí nastavená barva pro kreslení. Souřadnice 0,0 odpovídají levému hornímu rohu a kladně rostou směrem dolů a doprava.

Jednotlivé čáry a objekty můžeme formátovat přiřazením hodnot formátovacím parametrům ještě před začátkem samotného kreslení. Přiřazení barev čarám a výplni bude zobrazeno na dalším příkladu.

```
context.fillStyle   = '#00f'; // modrá barva výplně
context.strokeStyle = '#f00'; // červená barva čáry
context.lineWidth   = 4; // 4px tlustá čára

// Nakreslíme obdélníky.
context.fillRect(0, 0, 150, 50); // obdélník vyplněný barvou (bez obrysu)
context.strokeRect(0, 60, 150, 50); // obdélník kreslící pouze obrys
context.clearRect(30, 25, 90, 60); // vymaže zadanou obdélníkovou oblast
// na stejné oblasti vykreslíme obrys obdélníka
context.strokeRect(30, 25, 90, 60);
```

Na začátku kódu určíme, jakou barvu má mít výplň a obrys objektů. A dále určíme tloušťku obrysové čáry. Důležité je, že toto nastavení se projeví na všech objektech, které budeme od tohoto okamžiku dále kreslit. Výsledek je vidět na Obr. 1.

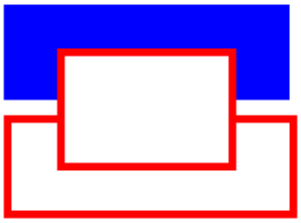


Obr. 1: Vykreslený obrázek na plátně canvas

Všimněte si, že spodní obdélník je oříznutý. To je způsobeno tím, že obrys se kreslí souměrně na všechny strany pomyslné čáry. Nekreslí se uvnitř obdélníku ani vně ale uprostřed. Abychom viděli celý obdélník, je třeba posunout druhý obrys o polovinu tloušťky čáry, což jsou 2 px doprava. Výsledný kód pro nakreslení druhého obdélníku bude vypadat následovně:

```
context.strokeRect(2, 60, 150, 50);
```

A nyní již vidíme žádaný výsledek. Spodní obdélník zřetelně o tloušťku čáry (4 px) širší. Což je dáno právě vykreslením obrysu uprostřed, kdy se z každé strany rozšíří obdélník o 2 px.



Obr. 2: Vykreslené obdélníky, kdy je spodní posunutý o 2 px doprava

Výše zmíněným postupem lze do plátna kreslit pouze základní objekty, jako jsou obdélníky kruhy, kruhové výseče, různé texty nebo vykreslování obrázků. Složitější objekty se kreslí pomocí cest (path). Kde se určí začátek cesty a pomocí čar se definuje tvar kresleného objektu. Cesty je možné kreslit jak pomocí rovných čar, tak třeba pomocí beziérovy křivky nebo kruhových výsečí. Nakreslení trojúhelníku je možné vidět níže:

```
context.beginPath();
// Začíná se v levém horním rohu
context.moveTo(10, 10); // souřadnice (x,y)
context.lineTo(100, 10);
context.lineTo(10, 100);
context.lineTo(10, 10);
// Nyní je určena cesta, který tvoří trojúhelník
context.fill(); // cestu vyplníme nastavenou barvou
context.stroke(); // a nakreslíme obrys
context.closePath(); // celou cestu nyní uzavřeme
```

Výsledek tohoto kódu je vidět na Obr. 3.

Obr. 3: Vykreslený trojúhleník na plátně canvas

Trojúhelník ani žádné jiné další tvary kromě výše zmíněných nelze kreslit jinak než pomocí cest. Tato metoda je však natolik univerzální, že je možné nakreslit prakticky jakýkoli tvar, který je potřeba.

Výše jsem nastínil možnosti objektu canvas, který se objeví i v některých dalších příkladech, nicméně se bude jednat pouze o určité možnosti. Souhrnně se tomuto elementu

již nebudu věnovat, protože tento element sám o sobě má tak široké možnosti uplatnění, že jen samotný popis tohoto jediného elementu by zabral celou další práci.

Příklad využití canvasu pro úpravu obrázku

Díky kombinaci tlačítka určeného pro nahrání souboru na server a elementu canvas jsem vytvořil demonstrační příklad, kde je možné vybrat obrázek na klientském počítači který, pak bude pomocí možností canvasu upraven do výsledné podoby a následně jej bude možné nahrát na webový server.

Obr. 4:Rozhraní demonstrativního příkladu v prohlížeči Google Chrome

Kliknutím na tlačítko vybrat soubor se zobrazí klasické dialogové okno pro výběr souboru. Po výběru souboru je ověřeno, zda se jedná o obrázek a pokud ano, pak je načten do pole canvas, kde je dále zpracováván. Nejprve je vybrána určitá oblast na obrázku, která je otočena vzhůru nohama a pak je na obrázek aplikován filtr sépie.

Obr. 5: Obrázek po úpravách zobrazený v oblasti canvas

Nakonec je možné jej pomocí odkazu „Nahraj na server“ asynchronně nahrát na webový server, kde bude uložena upravená verze souboru.

Obr. 6: Vybraný obrázek na klientském počítači (vlevo) a obrázek nahraný na serveru (vpravo)

Tento demonstrativní příklad provede všechny úkony automaticky za sebou je však možné vytvořit si vlastní webovou aplikaci, sloužící k úpravě obrázků, tak že na webový server již budou odesílány editované obrázky. Dříve bylo v podobných případech nutné na straně klienta jen zadat operaci, obrázek byl odeslán na webový server, kde byla operace provedena. A následně byl obrázek opět zobrazen pro další možnosti úpravy. Taková aplikace byla náročná na datový přenos tak i výpočetní výkon na straně serveru. Nyní je možné díky canvasu snížit oboje. Více v [Hassman Martin, 2009]

2 Technologie HTML5, JS a CSS3

Na samotném značkovacím jazyce HTML5 by nebylo nic výjimečného, kdyby neexistovali další technologie, které s webem souvisí. Těmi technologiemi jsou CSS a JavaScript, které ve spojení s HTML5 přináší na web nové možnosti práce s dokumentem.

2.1 Co je CSS?

Kaskádové styly (**Cascading Style Sheets** (**CSS**)) je strukturální jazyk popisující vzhled webových stránek. Jazyk byl navržen standardizační organizací W3C a byl vydán v roce 1997. Kaskádové styly lze použít jak pro formátování HTML dokumentů, tak pro formátování jakéhokoli dokumentu spadající do rodiny XML. V dalším textu budu však hovořit o kaskádových stylech pouze ve spojení s HTML dokumenty.

Abychom mohli říct, k čemu jsou kaskádové styly vhodné, je nejprve nutno říct, že každý dokument má svůj obsah a svou formu nebo formát či vzhled. V počátcích webových stránek se jak obsah, tak definice vzhledu psali do jediného dokumentu prostřednictvím různých HTML tagů a parametrů což při složitějším formátování způsobovalo nepřehlednost celého dokumentu. Kaskádové styly slouží primárně k oddělení obsahu a formátování dokumentu do dvou různých souborů nebo přinejmenším do dvou různých částí v HTML dokumentu. V samotném obsahu se tak objevují pouze HTML značky s případně jediným parametrem a vzhled tohoto elementu je definován právě v externím CSS dokumentu. To samozřejmě přináší značné výhody v momentě, kdy je třeba vzhled určitého elementu změnit. Dřív bylo nutné vyhledat v dokumentu všechny formátovací tagy a změnit všem parametry tak aby odpovídali novému vzhledu. Pokud web měl několik stránek, pak byla tato činnost velmi zdlouhavá. Při použití kaskádových stylů stačí změnit definici jen na jednom místě a upravený vzhled se projeví na všech stránkách, kterých se to týká.

CSS také přineslo nové možnosti formátování dokumentu, které dříve nebyly vůbec možné nebo jen obtížně realizovatelné. V průběhu let se formátovací možnosti CSS rozšiřovali což se děje i s příchodem CSS3.

Zápis CSS by se dal zjednodušeně popsat následně:

```
Selektory{
        Definice formátu 1;
        Definice formátu 2;
        …
}
```

Selektory

Selektory jako takové určují element v HTML dokumentu, kterého se dané definice týkají. Daný element může určovat pouze jeden selektor nebo celý řetězec selektorů, který povede až k danému elementu.

Základní selektory jsou následující:

- Název elementu – např. div, body, p, a apod.
- #idElementu – znak mřížky před textem určuje, že se jedná o element s daným atributem id

- .třídaElementu – znak tečky znamená že se definice vztahuje ke všem elementům s konkrétním atributem class

Tyto selektory je možné jakkoli kombinovat a určit tak konkrétní prvek nebo konkrétní prvky v dokumentu. Např. zápis

```
body div#obsah a.externi
```

všechny tagy a s atributem class="externi", které se nachází v tagu div s id="obsah" a ten se musí vyskytovat v tagu body. V HTML kódu se to týká těchto elementů.

```
<body>
<div id="obsah">
<a href="#">Tohoto odkazu se definice netýká</a>
<a href="#" class="externi">Tohoto odkazu se definice týká</a>
</div>
<a href="#" class="externi">Tohoto odkazu se definice netýká</a>
</body>
```

V tomto příkladě je možné přijmout určitá zjednodušení protože v jakýkoli zobrazitelný obsah se může nacházet pouze v tagu body, takže je možné selektor ukazující na tento tag odstranit. Druhým zjednodušením může být odstranění selektoru týkajícího se tagu div, protože hodnota atributu id musí být v celém HTML dokumentu jednoznačná, pokud tedy existuje element *div* s daným id, pak už nemůže existovat žádný jiný element se stejným *id*. To se však netýká atributu class, který se může v dokumentu vyskytovat duplicitně u různých elementů. Zkrácený zápis s fakticky stejným významem by vypadal následovně:

```
#obsah a.externi
```

Když jsou jednotlivé selektory oddělené mezerou, znamená to, že elementy vyhovující následujícímu selektoru musí být obsaženy v elementu nebo elementech, které odpovídají selektoru předcházejícímu. Takto je možné jít jakkoli do hloubky.

Kromě mezery je možné použít ještě znaku >, který určuje, že následující element se musí nacházet přímo v elementu, který odpovídá předcházejícímu selektoru. V případě mezery se nemusí nacházet přímo v daném elementu ale i jako potomci hlouběji ve struktuře.

A také je možné použít znaku + který říká, že následující element musí být v dokumentu za elementem uvedeným v definici jako předchozí.

Selektorů určující, o který element se jedná, je více. Zde jsou příklady některých z nich.

- element[atribut] – vybere elementy obsahující daný atribut
- element[atribut="hodnota"] – vybere elementy, pro které platí že daný atribut se rovná určité hodnotě
- element[atribut~="hodnota1 hodnota2"] – vybere elementy, pro které platí, že daný atribut obsahuje jednu z definovaných hodnot
- element:first-child – vybere element, který je prvním dítětem nadřazeného elementu
- element:first-line – vybere první řádek v elementu
- a další

Jednotlivé definice je také možné od sebe oddělit čárkou. Možnostmi formátování se zde nebudu zabývat, protože toto téma je natolik obsáhle, že by zabralo celou další práci. Pro zájemce je možné navštívit web W3C a přečíst si příslušnou normu. [Cascading Style Sheets (CSS) Snapshot 2010]

2.2 Nové selektory v CSS3

CSS3 na jednu stranu nepřináší žádný nový přístup ani další komplikované věci, jen rozšiřuje možnosti selektorů a přináší nové možnosti formátování. Píšu sice „jen nové možnosti“, ty jsou však docela zásadní. Dřív bylo sice možné dosáhnout stejného vizuálního efektu, jako dnes umožňuje CSS3 také, avšak mnohem komplikovanějšími přístupy. CSS3 přináší mnohá zjednodušení pro webové kodéry a designéry.

Vybírání elementů podle obsahu

Nově je možné vybírat elementy podle obsahu, který se v nich nachází. K tomu slouží pseoudotřída contains. Výběr elementu se provede následujícím způsobem.

```
p:contains("řetězec"); // vybere všechny odstavce obsahující text
„řetězec“
```

Rozšířené možnosti vybírání dle atributů

- element[atribut^="hodnota"] – vybere elementy, jejichž daný atribut začíná na určenou hodnotou
- element[atribut$="hodnota"] – vybere elementy, jejichž daný atribut končí na určenou hodnotou
- element[atribut*="hodnota"] – vybere elementy, jejichž daný atribut obsahuje určenou hodnotou

Selektory řídící se stromem dokumentu

- E:last-child – element E, který je posledním dítětem svého rodiče
- E:first-of-type – element E, který je prvním sourozencem svého typu
- E:last-of-type – element E, který je posledním sourozencem svého typu
- E:nth-child(n) – element E, který je n-tým dítětem svého rodiče
- E:nth-last-child(n) – element E, který je n-tým dítětem svého rodiče, počítáno odzadu
- E:nth-of-type(n) – element E, který je n-tým sourozencem svého typu
- E:nth-last-of-type(n) – element E, který je n-tým sourozencem svého typu, počítáno odzadu
- E:only-child – element E, který je jediným dítětem svého rodiče
- E:only-of-type – element E, který je jediným sourozencem svého typu

Seznam je převzat z [Dudek Jan, 2005]

Hlavní výhodou těchto pseoudoelementů je možnost dosadit za výraz *n* jakýkoli polynom, který pak bude určovat, o jaký prvek se jedná.

```
tr:nth-child(2n) td{
  /* nastavení buněk sudých řádků */
}
tr:nth-child(2n+1) td{
  /* nastavení buněk lichých řádků */
}
tr:first-child td{
  /* nastavení buněk prvního řádku tabulky */
}
tr:last-child td{
  /* nastavení buněk posledního řádku tabulky */
}
```

Příklad je převzat z [Dudek Jan 2005]

Existuje celá řada dalších možností jak vybírat elementy v dokumentu mnoho z nich však ještě nejsou stále implementovány v prohlížečích. Výše uvedený přehled jsem uvedl pro demonstraci toho, co bude díky CSS3 možné.

2.3 Nové vlastnosti v CSS3

CSS3 přináší nové vlastnosti, které zjednoduší formátování elementů. Nejpodstatnější z nich budou popsány níže. Mnohé z nich jsou již dnes, v roce 2011 použitelné.

Kulaté rohy

Zakulacené rohy, které jsou dnes k vidění prakticky na všech webových stránkách, nebylo možné klasicky definovat. Tento problém se nejčastěji řešil obrázky kulatých rohů naformátovaných tak aby vypadaly jako skutečné kulaté rohy stránky. V CSS3 stačí jednoduchý zápis formátu ve tvaru.

```
border-radius:5px;
```

Kde je určeno, že element bude mít všechny rohy zakulacené poloměrem 5 px nebo je možné zakulatit jednotlivé rohy samostatně.

```
border-top-left-radius: 2px;
border-top-right-radius: 3px;
border-bottom-right-radius: 4px;
border-bottom-left-radius: 5px;
```

Jak je z názvů patrné, tak roh vlevo nahoře bude mít poloměr 2 px vpravo nahoře 3 px vpravo dole 4 px a vlevo dole 5 px.

Barevné přechody

Stejně tak jako kulaté rohy, tak i barevné přechody jsou další prvek vyskytující se hojně na webových stránkách. A stejně tak jako kulaté rohy jsou všude řešeny pomocí obrázků. V CSS3 je možné definovat barevný přechod přímo v CSS zápisem:

```
background: linear-gradient(#EEFF99, #66EE33);
```

První barva bude nahoře a druhá barva bude dole. Mezitím se bude průběžně měnit. Toto je nejjednodušší definice gradientu. Je jí však možné i rozšířit pro více barev, kde bude přesně určeno kde barva začíná a kde končí.

```
background: linear-gradient(left, #1e5799 0%,#207cca 29%,#2989d8
50%,#207cca 78%,#7db9e8 100%);
```

První parametr určuje orientaci. Může obsahovat hodnoty top pro přechod shora dolů nebo left pro přechod zprava do leva. Každý další parametr určuje barvu a pozici, na které se bude daná barva zobrazovat. Jednotlivé pozice musí být uvedeny v procentech, protože není předem možné určit, jak velký bude element, na jehož pozadí bude přechod zobrazen.

Stíny pod elementy

Další možností je vykreslit pod libovolným blokovým elementem stín tak, že bude vypadat, jakoby na něj svítilo světlo. Stejně tak jako v předchozích případech tak i tentokrát se stíny v současné době řeší pomocí obrázků. V CSS3 se stín vykreslí následovně.

```
box-shadow: #666 10px 8px 5px;
```

První parametr určuje barvu stínu. Druhý posunutí stínu v ose x doprava, třetí posunutí v ose y dolů a čtvrtý sílu rozmazání stínu. Pokud je zadán druhý a třetí parametr záporný tak dojde k posunutí nahoru respektive doleva.

Stín pod textem

Stejně jako v případě stínu pod blokovým elementem je možné vykreslit stín pod textem. Definice je úplně stejná jako v předcházejícím případě jen název vlastnosti není box-shadow ale text-shadow.

```
text-shadow: #666 10px 8px 5px;
```

Význam jednotlivých parametrů se shotuje s předchozím případem.

Možnost použití vlastních fontů na stránce

Dříve bylo možné na webové stránce používat pouze fonty, které jsou nainstalované na klientském počítači. Pokud na klientském počítači font neexistoval, byl použit náhradní. Pokud webdesigner chtěl použít nějaké nestandardní písmo se zárukou toho, že bude web vypadat na všech zařízeních stejně, pak toto musel řešit obrázky.

V CSS3 je nyní možnost používat vlastní písma, jejichž definice jsou uložena na webovém serveru.

```
@font-face { /* určime si název písma a přiřadíme mu definici */
  font-family: SketchRockwell;
  src: url('SketchRockwell.ttf');
}
.my_CSS3_class { /* dále je pak možné písmo používat jako jakékoli jiné
*/
  font-family: SketchRockwell;
  font-size: 3.2em;
}
```

Transformace

V CSS3 jsou definovány 3 transformace, kterými můžete měnit zobrazení elementu. Jsou to:

1. Natočení
2. Zkosení
3. Přiblížení / Oddálení

V CSS se zapisují následovně:

```
transform: rotate(7deg); /* pootočí element o 7 stupňů */
transform: skew(-25deg); /* zkosí element o -25 stupňů */
transform: scale(0.5); /* zmenší element na 50% původní velikosti */
```

Jako další typ transformace můžeme použít také průhlednost, která však není řešení pomocí vlastnosti transform, ale má vlastní vlastnost opacity.

```
opacity:0.8;
/* element bude na 80% neprůhledný (čím menší číslo tím průhlednější) */
```

Další vlastnosti a jejich podpora

V CSS3 jsou definované i další vlastnosti ale jejich podpora je však v současné době velmi malá. Většina výše zmíněných vlastností není možné použít v Internet Exploreru 8 a v ostatních prohlížečích jsou přístupné většinou jen pomocí Andor prefixu (prefix před názvem vlastnosti pro Mozillu např -moz-) daného prohlížeče. Ve výsledku tak není kód tak jednoduchý jak je zde uváděno.

Pro použití stínů, přechodů a kulatých rohů však existuje snadné JavaScriptové řešení, které je k nalezení na [Css3PIE], kde stačí do stránky přilinkovat JavaScriptový soubor a tyto vlastnosti budou fungovat i v Internet Exploreru. CSS kód, který bude funkční, si můžete na této stránce také vygenerovat.

3 Pokročilé možnosti HTML5 a JavaScriptu

3.1 Aplikační keš - AppCache

AppCache je keš neboli paměť, do které jsou ukládány jednotlivé soubory webu tak, aby nemuseli být znova načteny v případě, že momentálně nemá webová aplikace přístup na internet. Do AppCache je možné ukládat HTML stránky, JavaScriptové soubory i jednotlivé obrázky. Zkrátka vše co je potřeba k běhu offline aplikace.

Jak dát webovému prohlížeči vědět co si má uložit?

Samotné ukládání se řídí podle pravidel zapsaných v manifestu, který je uložen na serveru. Manifest pak musí být připojen ke všem webovým stránkám, které aplikace využívá, jinak nebudou některé stránky v offline režimu dostupné. Manifest se připojuje ke stránce pomoci atributu manifest ve značce HTML.

```
<html manifest="test.manifest">
```

Soubor manifestu musí vracet mime typ text/cache-manifest, jinak nebude prohlížečem akceptován. Toho můžeme docílit mnoha způsoby avšak nejsnazší způsob na webovém serveru apache, je přidání souboru .htaccess do adresáře, kde je tento soubor uložen a vložit do něj následující řádek:

```
AddType text/cache-manifest .manifest
```

Všechny soubory s koncovkou .manifest tak budou vracet potřebnou hlavičku s mime typem text/cache-manifest.

Tím máme přilinkován soubor manifestu do webové stránky a zařídili jsme, že jej bude prohlížeč akceptovat. Další věc je co do manifestu napsat. Soubor manifestu může obsahovat 3 různé bloky, které řídí komunikaci se serverem a stahování jednotlivých souborů do keše. Jsou jimi bloky:

1. CACHE – Všechny soubory uvedené v tomto bloku budou uloženy do keše.
2. NETWORK – Soubory uložené v tomto bloku budou vždy stahované z internetu.
3. FALLBACK – Pro soubory v tomto bloku bude v případě offline verze použita definovaná náhrada.

Na prvním řádku manifestu se dále musí nacházet text CACHE MANIFEST, jinak nebude manifest funkční. Jednoduchý příklad manifestu je zobrazen níže:

```
CACHE MANIFEST
CACHE:
style.css
index.html
NETWORK:
style2.css
FALLBACK:
obrazek.jpg nahrada.png
```

V souboru manifestu je možné použít také komentáře, které musí být na samostatném řádku a musí začínat znakem #.

Všechny adresy musí být uvedeny relativně vůči souboru manifestu, ve kterém jsou zapsány nebo je možné samozřejmě použít i zápis absolutní adresy.

Výše uvedený manifest zařídí, že si prohlížeč do keše uloží soubory style.css, index.html a soubor.png. V případě offline režimu tak poběží stránka index.html s přilinkovaným stylem style.css. Přilinkovaný soubor style2.css nebude v offline režimu dostupný protože je dáno, že neměl být kešován a musí být vždy stažen z internetu. Zajímavé je to se souborem obrazek.jpg. Webový prohlížeč se jej nejprve pokusí stáhnout z webového serveru, což může trvat i několik sekund než vyprší případný timeout. A teprve pokud se mu jej nepodaří stáhnout, zobrazí místo něj obrázek nahrada.png, který si uložil dříve do keše.

Důležité je zmínit, že v případě kdy je přilinkován manifest, pak se veškerá síťová komunikace řídí podle definice, která je v něm uvedena. Pokud je ve webové stránce použit nějaký soubor, který není vůbec uveden v manifestu, pak se jeho stažení nepodaří a nebude pro webovou stránku vůbec existovat.

Dále je také nutno říct, že soubory uvedené v bloku NETWORK jsou označeny tak, že se s nimi má nakládat běžným způsobem, jako kdyby manifest neexistoval. Z toho plyne, že soubory uvedené v tomto bloku mohou být uložené v běžné lokální keši, pokud prohlížeč usoudí, že je to výhodnější. Ukládání v běžné keši je možné řídit pomocí http hlaviček Cache-Control nebo expires.

Pokud je třeba webovou keš obnovit je to možné dvěma způsoby:

1. Obnovit webovou stránku
2. Vyvolat obnovu pomocí JavaScriptu

V obou případech je obnova podmíněna změnou souboru s manifestem. Nejprve si totiž prohlížeč ověří, zda se soubor změnil a teprve pokud ano, pak dojde k obnovení stávající keše.

První způsob není třeba nijak zvlášť rozvádět. Aktuálnost se vždy zkontroluje v případě kdy je stránka obnovena. Druhá možnost využívá API, pomocí kterého může JavaScriptová aplikace řídit kontrolu keše a její použití. Po zavolání následující metody dojde k ověření, zda byl soubor manifestu změněn a pokud ano, pak k opětovnému stažení všech souborů v manifestu do aplikační keše.

```
window.applicationCache.update()
```

Důležité je však říct, že i když je nová keš už stažena, tak stále nedojde k jejímu použití. K tomu by došlo až při dalším obnovení stránky. Použití nove keše je ale možné vynutit zavoláním metody *swapCache*, kterou je vhodné rovnou navázat na událost *onupdateready*, která se vyvolá po stažení všech souborů v manifestu. Následující kód zařídí, že ihned po stažení nové keše dojde jejímu použití.

```
addEventListener("onupdateready", function(){
        window.applicationCache.swapCache();
}, false);
```

Pomocí aplikační keše a s použitím nějaké metody ukládání dat u klienta, které budou zmíněny dále, je možné vytvářet offline webové aplikace pouze s použitím HTML, CSS a JavaScriptu. Takové aplikace poběží v jakémkoli webovém prohlížeči, který implementuje nové možnosti HTML5.

3.2 WebStorages

WebStorages je jednoduché uložiště typu klíč-hodnota, do kterého si může aplikace ukládat svá data ve webovém prohlížeči. Tyto data jsou pak vždy přístupná ze všech stránek dané domény. Velikost tohoto úložiště je teoreticky neomezená. Prakticky bude záležet na konkrétní implementaci v prohlížečích.

Specifikace definuje dva typy úložiště:

1. **localStorage** – Trvalé úložiště, kde jsou data uložená trvale a jejich případné odstranění je plně v režii aplikace nebo uživatele.
2. **sessionStorage** – Úložiště dostupné během sezení. Uložená data setrvají na disku jen během sezení. Kdy je sezení ukončeno a data smazána záleží pouze na implementaci v konkrétním prohlížeči.

Jednotlivá úložiště jsou definována přímo nad objektem window. Přístup k nim je tak možný buď přes rodičovský objekt, nebo je možné jej i vynechat a přistupovat k úložišti přímo.

```
Window.localStorage // přístup přes rodiče
Windows.sessionStorage
```

V dalších ukázkách bude použit přímý přístup, protože v JavaScriptu je obecně možné rodičovský element Windows vždy vynechat.

Obě dvě úložiště mají stejné metody a parametry jediný rozdíl mezi nimi je v době po jakou budou data uložena, proto se dále budu zabývat pouze jedním úložištěm a tím je localStorage. Úložiště obsahují tyto metody:

- getItem(key) – Vrátí hodnotu odpovídající danému klíči.
- setItem(key, value) – Uloží pod zadaný klíč danou hodnotu.
- removeItem(key) – Odstraní záznam s daným klíčem.
- clear() – Odstraní všechny data v úložišti.
- key(index) – Vrátí klíč odpovídající danému číselnému indexu.

V úložišti jsou data automaticky číslovány od nuly. Stejně tak je možné zjistit počet všech uložených záznamů v úložišti pomocí parametru length a díky metodě key je tak možné procházet všechny položky v úložišti pomocí cyklu.

Pro přístup k záznamům je možné používat zjednodušený zápis, jakoby se jednalo o pole s daným klíčem. Metody setItem a getItem jsou volány automaticky na pozadí. Následující kód vykoná v obou případech to samé a to že uloží do úložiště hodnotu pod daným klíčem.

```
localStorage.setItem("klíč","hodnota");
localStorage["klíč"] = "hodnota";
```

Druhý přístup je elegantnější ale pro programátora neznající webStorages není zcela zřejmé, co tento kód ve skutečnosti udělá, protože to vypadá, jako že se ukládají data pouze do nějakého pole a ne do trvalého úložiště.

Stejně tak si můžeme data i vrátit.

```
var data1 = localStorage.getItem("klíč"); // vrátí řetězec "hodnota"
var data2 = localStorage["klíč"]; // vrátí stejný řetězec "hodnota"
```

Pro následné odstranění musíme však vždy použít metodu removeItem. Pro tento případ žádné zjednodušení neexistuje. Stejně tak další metody se volají standardně nad daným objektem, ať už se jedná o localStorage nebo sessionStroage.

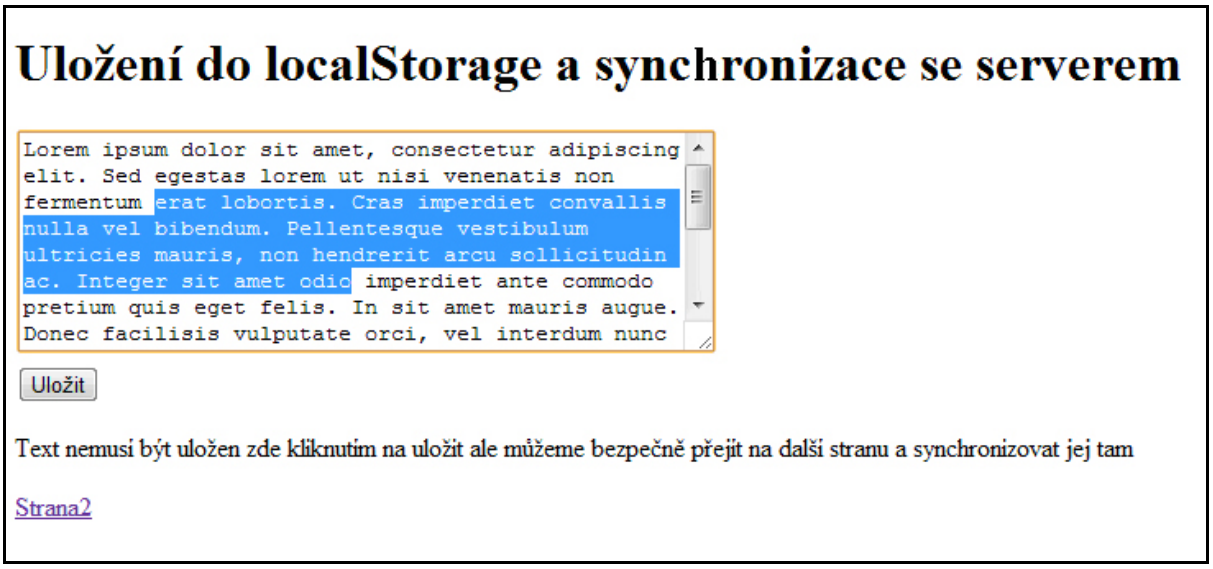
V případě změny v úložišti je vyvolaná nad objektem Windows popřípadě dokument událost **storage**. Tuto událost můžeme standardně obsloužit námi naprogramovanou funkcí. Důležité je, že v objektu události se nachází atributy, které přímo souvisí s měněným záznamem. Jsou to:

- key – klíč položky, které se operace týkala
- oldValue – původní hodnota, nebo hodnota null, pokud byla položka vytvořena nově
- newValue – nová hodnota položky
- url – URL stránky se skriptem, který vyvolal událost
- storageArea – oblast, v níž došlo ke změně (local/session storage)

V současné době však obsluha události storage není v žádném prohlížeči zcela funkčně implementována, není ji tak zatím možné spolehlivě použít.

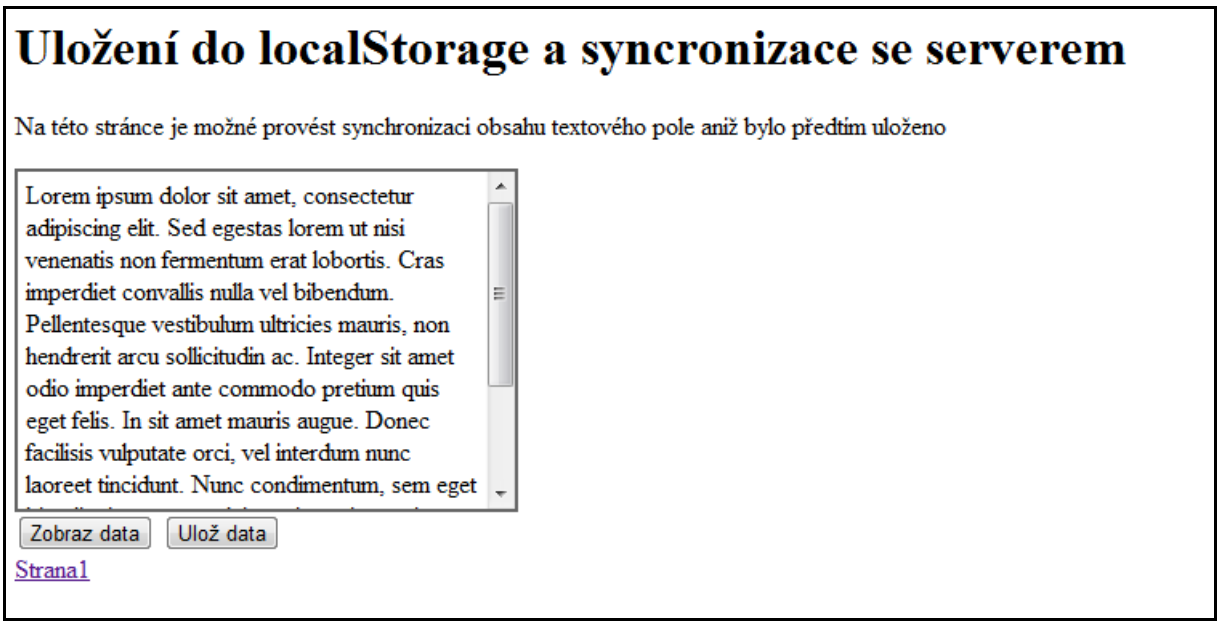
Příklad využití webStorages

K oblasti webStorages jsem vytvořil testovací aplikaci, s jednoduchým formulářem, kde je možné psát text. Tento text se spolu s pozicí kurzoru ukládá do localStorage pro případ ztráty připojení k internetu nebo pro možnost přechodu na jinou webovou stránku a návratu zpět k rozdělané práci bez ztráty dat. Tuto ukázku můžete vidět na Obr.7.



Obr. 7: Strana1 s textovým polem a vybraným textem

Na druhé stránce příkladu je možné zobrazit si obsah, který byl zapsán do textového o pole a uložit jej na web aniž by se uživatel musel vracet zpět na původní stránku.



Obr. 8: Strana2 po zobrazení dat z textového pole s minulé stránky

Po návratu zpět na stránku 1 se do textového pole uloží data uložená v úložišti a také se podle uložených informací nastaví kurzor pro psaní. Výsledek tak bude vypadat naprosto stejně jako na Obr. 7 Je tak možné pokračovat v činnosti přesně na místě, kde uživatel předtím skončil. V případě ztráty připojení a jeho opětovném obnovení dojde automaticky k uložení dat na server, ať už se uživatel nachází na jakékoli stránce.

Pro zjištění, zda má webový prohlížeč přístup k internetu slouží atribut objektu navigátor. **window.navigator.online** vracející hodnoty true nebo false. Aby se tato hodnota nemusela periodicky ověřovat, jsou také při změně stavu volány nad objektem document.body následující události. Při přechodu ze stavu offline do stavu online je vyvolána událost „**online**“. Při přechodu do stavu offline je vyvolána událost „**offline**“. Tyto události je pak možné standardně zaznamenat a patřičným způsobem na ně reagovat.

3.3 Databáze v prohlížečích a Filesystém

S HTML5 přichází také možnost využívat relační databáze přímo ve webových prohlížečích nicméně je tato oblast dalším místem sporů mezi prohlížeči, kteří se nemohou dohodnout jaký přístup k databázi zvolit.

První možnosti je Web Database, která funguje jako SQL databázový klient a interně pracuje na bázi sqlLite. Databázi se zašle SQL ta jej zpracuje, vykoná, co jí bylo zadáno, a vrátí výsledek. Jedná se tedy o běžnou SQL relační databázi, která jen vrací výsledky dle

požadavků. Tento přístup podporuje Chrome, Safari a Opera a v současné době, v roce 2012, je již funkční. Proti tomuto řešení jsou však tvůrci webového prohlížeče Firefox, kteří prosazují objektovou databázi nazvanou IndexedDB. Ta umožňuje ukládání celých objektů do databáze a jejich snadné procházení na úrovni JavaScriptu. Navíc tuto možnost považují i vývojáři Internet Exploreru jako perspektivnější.

Vzhledem k tomu, že v současné době existují pouze implementace relačních databází na principu SQL, které nejsou a zřejmě nikdy nebudou podporovány v majoritních prohlížečích Internet Exploreru a Mozille Firefox, nebudu se jimi prozatím dále zabývat.

Souborový systém

V HTML5 existuje specifikace umožňující ukládání souborů na straně klienta ve vyhrazené části filesystému sloužící výhradně pro ukládání souborů pomocí JavaScriptu. S těmito uloženými soubory je pak možné dále pracovat a nahrávat jej na server. Bohužel stejně jako v případě databází není Filesystem API podporováno v žádném z majoritních prohlížečů. Jediná trochu funkční implementace je v prohlížeči Google Chrome. Proto se tímto tématem stejně jako v případě databází nebudu dále zabývat.

3.4 Dataset - Zápis vlastních atributů k elementům

V současné době neexistuje žádný standardizovaný způsob jak k nějakému elementu v DOMu přidat vlastní atributy tak, aby mohly být následně zpracovány v JavaScriptu. Nejčastěji se to řešilo přidáním potřebných hodnot do atributů rel nebo class, které pro to nejsou určeny. S příchodem HTML5 přichází i standardizovaný způsob jak k jakémukoli elementu přiřadit atribut s vlastním názvem, který může být dále zpracováván pomocí JavaScriptu. Tím jsou uživatelské datové atributy. Všechny tyto atributy musí začínat prefixem *data-* a měly by obsahovat pouze malá písmena. Hodnota těchto atributů může být jakýkoli řetězec.

Pro přístup nebo nastavení k těmto elementům je možné použít standardní metody, které je možné používat již dnes.

```
var element = getElementById('idElementu');
element.setAttribute('data-muj-atribut', 'moje hodnota');
element.getAttribute('data-muj-atribut'); // vrátí řetězec 'moje hodnota'
```

```
element.removeAttribute('data-muj-atribut'); // odstraní atribut z
elmentu
```

Tento způsob však není pro datové atributy určen. Specifikace říká, že k datovým atributům je možné přistupovat pomocí objektu dataset. Pro představení funkčnosti datasetu je nejprve nutné definovat kousek HTML kódu.

```
<div id="idElementu" data-muj-atribut="moje hodnota">
 … nějaký obsah …
</div>
```

Tento HTML kód může být zpracován JavaScriptem v minulém příkladu. Nyní si však ukážeme přístup pomocí datasetu.

```
var element = getElementById('idElementu');
element.dataset.mujAttribut; // obsahuje řetězec "moje hodnota"
element.dataset.mujAttribut = 'změněná hodnota'; // změní hodnotu
atributu
element.dataset.mujAttribut = null; // Odstraní atribut s HTML DOMu
```

Všimněte si, že k atributu uvedeném v HTML kódu „data-muj-atribut“ je pomocí datasetu přistupováno pomocí tzv. CamelCase. U všech názvů atributů obsahující prefix *data-*, je tento prefix odstraněn a každé písmeno za pomlčkou je změněno z malého na velké. Z toho důvodu je také doporučeno používat v data atributech pouze malá písmena.

Datasety jsou standardní možností jak si k jednotlivým elementům na webové stránce ukládat libovolné informace do vlastních atributů. Tyto atributy by měli sloužit pouze pro účely dané aplikace a neměli by být nijak jinak využity ať už externí aplikací nebo pro formátování dokumentu. K tomu jsou určeny jiné možnosti.

V současné době není přístup pomocí datasetu příliš podporován. Je proto nutné používat standardní metody setAttribute a getAttribute. Důležité však je, že tento standart byl již přijat a nehrozí tak dopředná nekompatibilita.

3.5 WebWorkers – vícevláknový JavaScript

JavaScript jako takový běží vždy jen v jednom vlákně, to znamená, že kód, který se má vykonávat je zpracováván řádek po řádku. Dokud se nevykoná jedna činnost, nemůže se začít vykonávat činnost další. Ve většině případů je tento přístup dostačující, problém však

nastává v okamžiku, kdy je třeba provést nějakou výpočetně náročnou operaci. V tom okamžiku celý webový prohlížeč zatuhne a čeká se, dokud tato operace nebude dokončena. Teprve po dokončení této operace je možné vykonávat další činnost a prohlížeč se opět rozběhne. To se snaží webWorkers obejít. Jedná se v podstatě o snahu přidat do JavaScriptu možnost, jak něco zpracovávat na pozadí aby nedošlo k zastavení vykonávání hlavní činnosti.

WebWorkers však mají i svá omezení. Tím, že běží ve vlastním vlákně, tak není možné přistupovat k žádnému objektu z hlavního vlákna. To se týká zejména přístupu k objektům DOMu, ke kterým nemá worker vůbec přístup. Stejně tak, je omezen přístup k objektu location kdy není možné měnit jeho hodnoty a slouží pouze pro čtení. Výměna informací mezi hlavním a webWorkerovým vláknem je možná pouze pomocí zpráv, které jsou zpracovávány pomocí událostí.

Níže je ukázán jednoduchý příklad, jak s webWorkerem pracovat. Mějme například následující kód, ve kterém se nachází časově náročná operace. Ta způsobí čekání, dokud není proces ukončen.

```
<script>
var i;
var cyklu = 500000000;
for(i = 0; i < cyklu; i++){} // časově náročná operace
// čeká se dokud cyklus nedokončí svou práci
alert(i); // teprve pak se vyvolá okno s výsledkem
</script>
```

Časově náročná operace je zde pro jednoduchost reprezentována pouze cyklem s velkým počtem opakování ale může se jednat o jakoukoli jinou operaci. Nyní tento kód upravíme pro použití s webWorkerem. V hlavním okně budeme mít tento kód:

```
var worker = new Worker('worker.js'); // vytvoření nového workeru
var cyklu = 500000000;
// funkce, která zpracuje událost po zavoláni postMessage s vlákna
workeru
worker.onmessage = function (e){ // zpracování dat s workeru
  alert(e.data); // data přijatá od workeru
};
worker.postMessage(cyklu);
// kód normálně pokračuje bez zastavení běhu
```

A v souboru worker.js tento:

```
// funkce, která zpracuje událost po zavoláni postMessage s hlavního okna
onmessage = function(e){
        var i;
        var cyklu = e.data; // data, která jsme zaslali workeru
        for(i = 0; i < cyklu; i++){} // časově náročná operace
        postMessage(i); // odeslání výsledku do hlavního vlákna
}
```

Takto máme naprogramované řešení stejné úlohy ve více vláknech. Časově náročná operace je zpracovávana ve vedlejším vlákně a nebrzdí tak vlákno hlavního procesu. Díky tomu se prohlížeč nezasekne a bude pokračovat dál ve své činnosti. Práce ve vedlejším vlákně začne v okamžiku zavolání worker.postMessage(data) na což je následně zavolána funkce onmessage(event) v souboru worker.js, která již běží ve vedlejším vlákně. Zde je proveden skutečný výpočet. V okamžiku, kdy bude operace ve vedlejším vlákně zpracována, zavolá se funkce postMessage, která odešle výsledek zpět do hlavního vlákna. Zde je odchycena stejnojmennou funkcí onmessage, definovanou nad objektem workeru, ve které je zobrazen výsledek pomocí dialogového okna alert. Důležité je si všimnout, že přeposílaná data se nachází v objektu event, který je předáván mezi funkcemi v parametru data.

WebWorkers je dobrý pomocník v případě zpracovávání náročných výpočtů. V HTML5 se může jednat například o zpracování velkých obrázků nebo jakýchkoli jiných souborů, u kterých se dá předpokládat, že jsou časově zdlouhavé. Není však možné měnit strukturu DOMu ani provádět různá přesměrování a jiné akce, které mohou ovlivnit hlavní vlákno.

3.6 Technologie Drag and Drop

Drag and drop je technika, při níž je objekt uchopen a je možné ho přetáhnout na jiné místo a tam ho pustit. Tato technika se běžně používá v různých uživatelských rozhraních. V HTML4 se tato technika řešila nejčastěji kombinací, při níž se hlídalo, zda je myš zmáčknuta. Po té se animoval přesun během tažení objektu a nakonec se ošetřila závěrečná fáze, kdy byla myš puštěna. Takže přesun v rámci stránky se dal realizovat i dříve a nebylo to ani příliš komplikované.

HTML5 však přináší standard pro přesun objektů. Co se týče přesunu v rámci jedné webové stránky, je zde jediné zjednodušení a to, že se nemusí animovat přesouvaný element. Celou animaci provede prohlížeč za nás. Pro Drag-n-drop přesun jsou definovány tyto události a při úspěšném přetažení jsou volány v následujícím pořadí:

1. dragstart – Je vyvolána nad taženým elementem při začátku přesunu.
2. dragenter – Je vyvolána nad kontejnerem, který je definován pro možnost položení elementu v okamžiku, kdy je tažený element přetáhnut nad něj.
3. dragover – Je volána při pohybu taženého elementu nad kontejnerem.
4. drop – Je vyvolána nad kontejnerem v okamžiku puštění taženého elementu.
5. dragend – Je vyvolána vždy po konci přetažení, ať už byl element úspěšně přesunut nebo bylo přetažení zrušeno. Slouží pro uklizení všech prostředků vytvořených pro přetažení elementu.

Z výše uvedených událostí je zřejmé, že musí existovat 2 typy elementů. První je element, který bude možné přetáhnout a druhý, který bude sloužit pro pokládání. Na oba se pak musí navázat jednotlivé události.

```
<div id="kontejner" ondragover="return over(event)" ondrop="return
drop(event)" ondragenter="return enter(event)">
Prostor pro položení elementu
</div>
<div id="tazeny" draggable=true ondragstart="return start(event)"
ondragend="return end(event)">
Tažený element
</div>
```

Element, který bude možné přetáhnout, musí mít atribut *draggable*, tím se aktivuje možnost přesunu. Pokud však chceme provést nějaké akce na základně vlastního přesunu, je třeba implementovat i další metody, které musíme obsloužit. Těmi jsou *ondragstart* a *ondragend*. Funkce pro obsluhu jsou definovány níže:

```
function start(e){
     e.dataTransfer.setData("Text", e.target.id);
}
function end(){
        e.dataTransfer.clearData();
}
```

Ve funkci start můžeme přidat do objektu *dataTransfer* informace, které budou dostupná během celého přetažení. První parametr definuje *mime* typ dat a druhý samotná data. Je možné poskytnout více dat s různým *mime* typem. V našem případě přidávám do *dataTransferu* textová data obsahující id elementu, který bude tažený.

Ve funkci end je třeba uklidit vše, co bylo vytvořeno během tažení elementu. V našem případě jen vyprázdním objekt *dataTransfer*.

Nyní si definujeme funkce, které budou provádět obsluhu kontejneru.

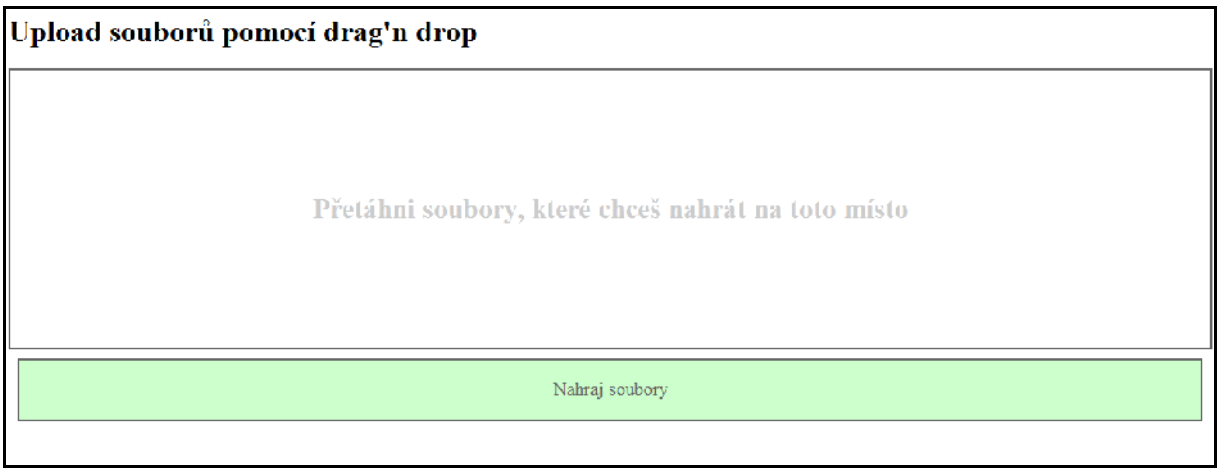
```
function enter(){
        //nemusí se provést nic
}
function over(e) {
        // pouze zastavíme předávání události do dalších elementů
        // není třeba provádět další akce
        e.preventDefault();
        return false;
}
function drop(e){
                // přečteme id s objektu dataTransfer
        var id = e.dataTransfer.getData("Text");
                // najedem element s daným id v dokumentu
        var el = document.getElementById(id);
                // vložíme jej do kontejneru nad, kterým byl puštěn
        e.target.appendChild(el);
}
```

Veškerá obsluha chování co se má stát po spuštění je ve funkci drop. Zde je ukázáno, jak tažený element přesunout z původního místa na stránce a vložit jej do kontejneru pro spuštění.

Podobným způsobem bylo možné elementy přesouvat po stránce již dříve. Hlavní výhoda standardního drag-n-drop API je však v možnosti přesouvat jednotlivé elementy i napříč různými okny prohlížeče nebo je dokonce možné přesouvat soubory z klientského počítače do kontejneru pro položení a tam je dále zpracovat. Pro tento případ jsem vytvořil další příklad, který tyto možnosti demonstruje.

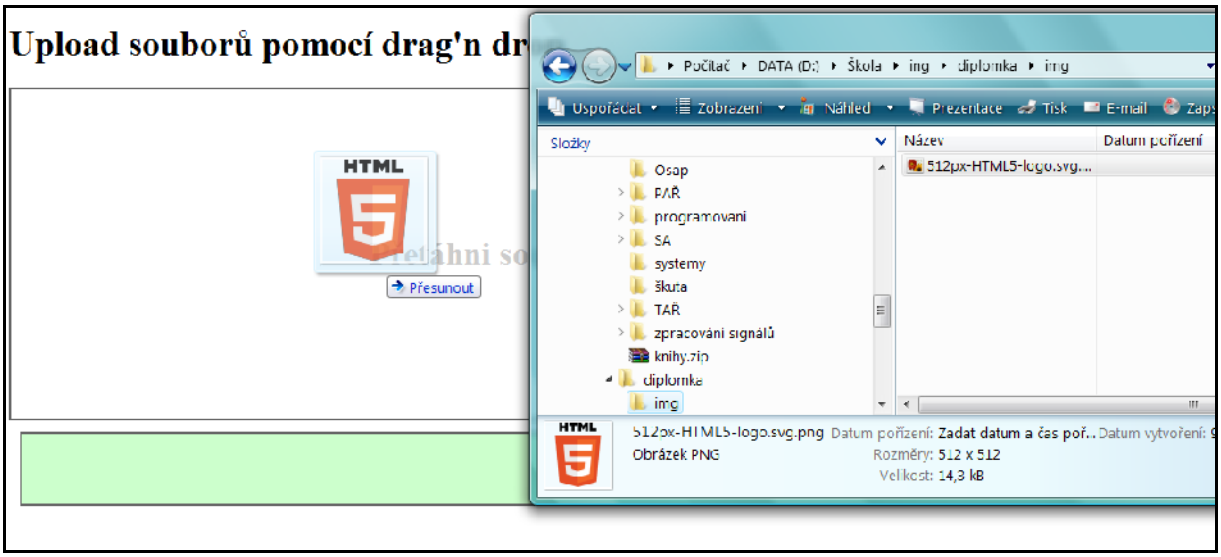
Příklad uploadu souborů pomocí přetažení do okna prohlížeče

Na začátku je jednoduchá stránka, na které je definován kontejner pro položení objektu a velké tlačítko pro asynchronní upload. Tato stránka je formátována pomocí CSS a nad pokládacím kontejnerem jsou definovány všechny výše uvedené události obsluhy. Jediný rozdíl je ve funkci drop, která je komplikovanější, protože se nestará pouze o přesun elementu v rámci stránky, ale musí obsloužit i přesun celého souboru.



Obr. 9: Úvodní obrazovka demonstračního příkladu

Soubor může být do webového prohlížeče přetažen například z uživatelské plochy nebo jakékoli jiné složky. Stačí jej jen chytit a přetáhnout do okna prohlížeče nad daný kontejner. Automaticky se zobrazí animace informující o tom, že soubor je možné přesunout do prohlížeče.



Obr. 10: Přetažení souboru z klientského počítače do webové stránky

V okamžiku, kdy je soubor spuštěn nad kontejner, tak je dále zpracován. Dojde k ověření, jestli se jedná o obrázek a pokud ano, pak je zobrazena odpovídající položka, která jej reprezentuje. Ta se skládá ze samotného obrázku, který je načten z klientského počítače a z názvu souboru. V tomto okamžiku ještě stále nedošlo k žádné komunikaci se serverem a vše je zpracováváno ve webovém prohlížeči.

Obr. 11: Zpracovaný obrázek připravený pro nahrání na server

Po kliknutí na tlačítko „Nahraj soubory“, jsou všechny soubory, které jsou momentálně zobrazeny na stránce, nahrány na server. Souborů je možné vložit libovolné množství a odeslány budou všechny najednou. Během nahrávání je zobrazen jednoduchý ukazatel, který informuje, kolik procent dat již bylo odesláno.

Tento příklad demonstruje jak vložit soubor do webové stránky s možností jej dále zpracovat a nahrát na server. Dále by ho bylo možné rozšířit například o možnost editace obrázku, možnost zrušení načteného souboru pokud jej již nechci nahrát a o mnohé jiné vylepšení, tak aby byl uživatelsky přívětivější.

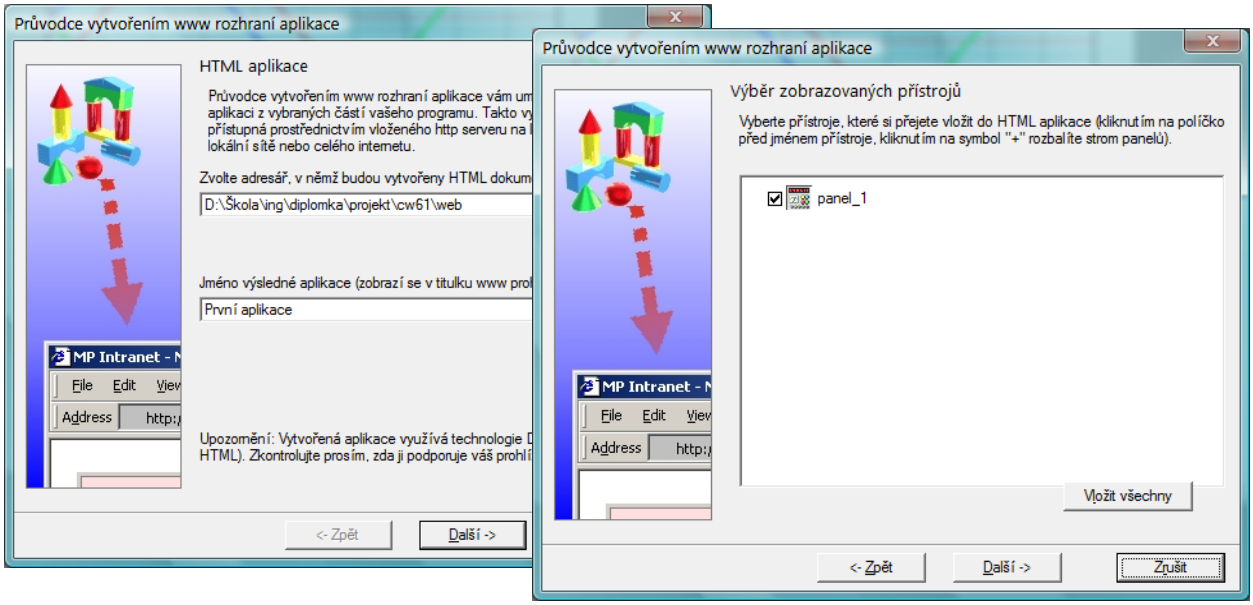
4 Vytváření webového rozhraní v prostředí ControlWeb 6.1

Program ControlWeb slouží k vytváření vizualizačních prostředí pro zobrazení a ovládání nejrůznějších úloh. Součástí tohoto programu je také možnost vytvoření webového serveru, který umožňuje obsluhu procesu z celého internetu pomocí webového rozhraní. Webové rozhraní je možné vytvářet manuálně avšak tento způsob je velice

zdlouhavý a může při něm docházet k chybám. Mnohem častější je využití průvodce pro vytvoření webového rozhraní, pomocí něhož je možné vytvořit funkční web na pár kliknutí.

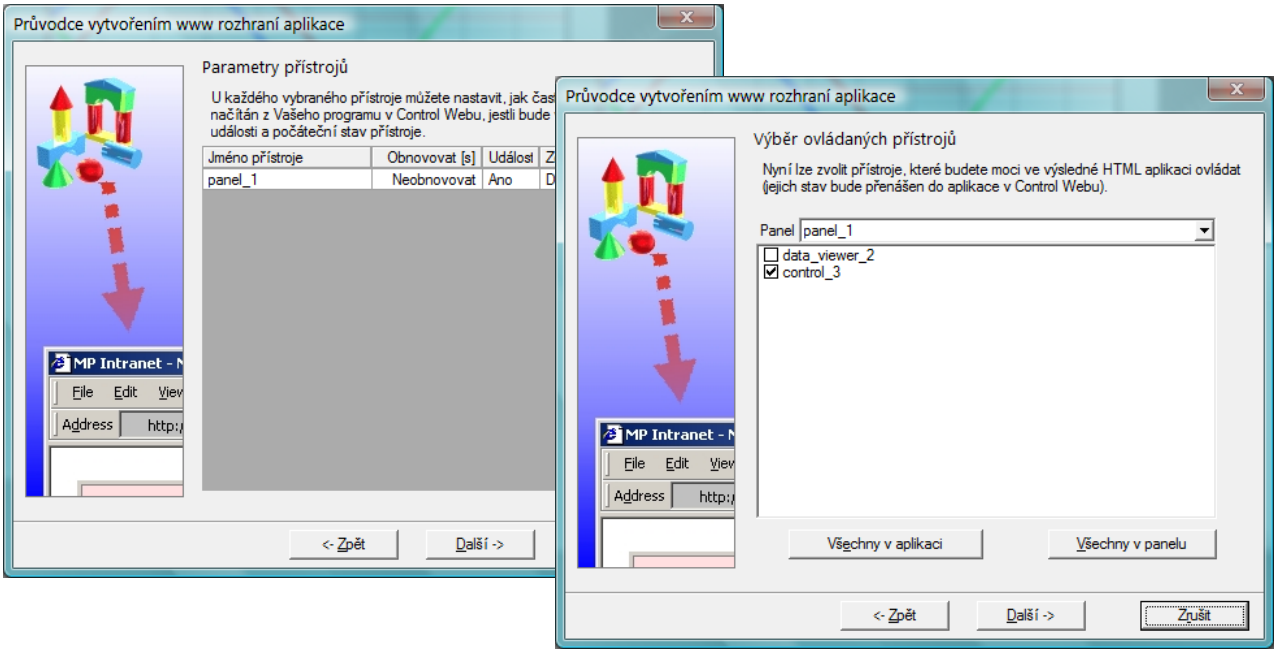
Výsledný web pak vypadá prakticky stejně jako rozhraní v počítači a je tvořen pomocí obrázků jednotlivých ovládacích a vizualizačních prvků. Webová stránka se pravidelně obnovuje v zadaném intervalu a všechny zobrazené prvky jsou stažené a překreslené novými. Zde se však nachází problém, který budu dále s pomocí HTML5 řešit.

Při vytváření webu je nejprve nutné uvést adresář, kde budou všechny webové prvky uloženy a také název aplikace. V dalším kroku jste vyzváni k výběru přístrojů, které budou zobrazeny.



Obr. 12: Průvodce pro vytvoření webové aplikace – nastavení adresáře a zobrazené panely

Když vyberete vše potřebné, pak je už jen nutné nastavit, jak často se má webová aplikace obnovovat, jaké prvky mají být na webu interaktivní a jaké se mají pouze zobrazit.



Obr. 13: Průvodce pro vytvoření webové aplikace – nastavení obnovování a interaktivních prvků

Poté co vyberete všechna nastavení a stisknete dokončit je web už hotov. Při spuštěné aplikaci je dostupný klasický webový server, který si můžete na lokálním počítači zobrazit na adrese 127.0.0.1.

V nastavení http komponenty je pak možné provádět další nastavení, avšak toto nebude nutné. Cílem této práce je integrovat nové možnosti HTML5 do ControlWebu s minimálním úsilím ze strany tvůrce aplikace.

4.1 Využití HTML5 při vytváření webového rozhraní

Změna vizualizace spočívá v nahrazení statického obrázku a pravidelného obnovování celé webové stránky za dynamické načítání dat na pozadí a vykreslování grafů, ukazatelů a zobrazovačů dynamicky pomocí HTML5. Výsledkem je, že se grafy a ukazatele mohou obnovovat mnohem rychleji a může dojít k rychlejší reakci obsluhy. S uživatelského hlediska je také mnohem lepší, když se grafy jen překreslují na obrazovce, než když se musí vždy pravidelně obnovit celá stránka.

4.2 Výhody použití HTML5

Obrázek je z datového hlediska poměrně velký a při překreslování je jej nutné vždy stáhnout. Jestliže se na webové stránce vyskytuje velké množství přístrojů = obrázků, pak

je nutné přes internet přenést poměrně velké množství dat. Nahrazením obrázků pouze daty, které se nějakým způsobem vykreslí až na straně prohlížeče, dojde k několikanásobnému snížení datového toku a sníží se tak nároky na síťovou komunikaci.

Druhým pohledem je uživatelská přívětivost dané aplikace. HTML5 umožňuje asynchronní načítání dat ve webovém prohlížeči, kdy pro načtení nových dat není nutné obnovit webovou stránku [AJAX]. Nově stažená data jsou dynamicky zobrazena a webový prohlížeč nemusí vykreslovat celou webovou stránku od začátku. Uživatel díky tomu neztrácí koncentraci na danou úlohu a pracuje efektivněji.

Třetí výhoda vyplývá z té první. Jestliže je datový tok několikanásobně menší, pak je možné zvýšit frekvenci odesílání a obnovování uživatelského rozhraní. Při využití klasického rozhraní generovaného ControlWebem je možné obnovovat webovou stránku pouze jednou za sekundu. Načítání webové stránky je v tomto případě pomalé a uživatel je každou sekundu přerušen obnovením webu. Při využití dynamického vykreslování a asynchronního načítání je možné obnovovat elementy až 10x za sekundu bez jakékoli nutnosti obnovit celou webovou stránku.

4.3 Nahrazení obrázkového rozhraní dynamickým HTML5

Při mé další práci jsem se rozhodl nahrazovat pouze prvky, které indikují stav daného procesu. Typicky se jedná o grafy, různé ukazatele, displeje apod. Ovládací prvky jako různé potenciometry, tlačítka apod. nahrazovat nebudu a při jejich použití bude ponechána výchozí funkčnost dána průvodcem při jejich vytváření. Nahrazení těchto prvků by bylo mnohem náročnější pro implementaci ve vytvářených úlohách a vynaložená práce by přinášela pouze malý užitek. Navíc tyto prvky nezobrazují stav úlohy a není nutné je pravidelně načítat.

Pro vykreslování grafů a ukazatelů jsem se rozhodl využít existující knihovnu RGraph [The RGraph documentation and examples], která je zdarma pro nekomerční využití. Tato knihovna využívá elementu canvas pro vykreslení grafů a zapouzdřuje rozdíly mezi webovými prohlížeči do jediné knihovny [Hassman Martin, 2009]. Součástí této knihovny je také podpora canvas elementu v Internet Exploreru, který je simulován pomocí VML grafiky. Pro Internet Explorer verze 9 již tato simulace není zapotřebí, protože je elment canvas plně podporován.

Knihovna RGraph podporuje tyto webové prohlížeče:

- Mozilla Firefox 3.0+
- Google Chrome 1+
- Apple Safari 3+
- Opera 9.5+
- Microsoft Internet Explorer 7+
- iPhone (text support from iOS v4+)
- iPad (text support from iOS v4.2+)

Výsledkem je, že grafy této knihovny fungují v 99% používaných webových prohlížečů.

5 Program běžící v ControlWebu

Uživatel, který toto grafické rozhraní bude používat nepotřebuje znát jeho vnitřní implementaci. Pro správnou funkčnost stačí zapsat do procedur _show a _reload elementy, které se mají ve webovém prohlížeči vykreslit. K tomu se využívají 3 procedury, kterým se jen předají dané parametry a o další komunikaci se postará hlavní aplikace. Uživatel je tak zcela odstíněn od jakékoli komunikace mezi ControlWebem a webovým rozhraním.

Procedury, které se starají o předání správných parametrů jsou safe(…), show(…) a setParam(…). Ty se také starají o ukládání dat do bufferu a generování JSON objektu, který je dále předán do aplikace ve webovém prohlížeči.

5.1 Ukládání dat v ControlWebu

Pro běh aplikace v ControlWebu je nezbytné průběžné ukládání dat, které se teprva až při požadavku na obnovení odešlou do webového prohlížeče. V ControlWebu je proto vytvořen speciální datový element prom_grafy, ve kterém jsou uloženy následující skalární proměnné.

```
    names : array[ 0..1000000 ] of string; // slouží pro uložení názvu
klíčů
    values : array[ 0..1000000 ] of real {init_value = 0}; // slouží pro
uložení hodnot
    times : array[ 0..1000000 ] of real {init_value = 0}; // časy kdy
byly zaznamenány hodnoty
    init_cas : cardinal; // čas kdy byla spuštěna aplikace
```

```
    pocet_hodnot : longcard; // počet uložených hodnot
    pocet_el : cardinal; // počet elementů, které jsou generované v proc.
_reload
    je_nacten : boolean; // pomocná proměnná na startu je false po prvním
zavolání _reload je true
    vystupni_index : real; // počítá počet zavolání show()
    html_text : string; // textový řetězec, do kterého se generují částí
JSON objektu v proceduře show
    param_text : string; // textový řetězec, do kterého se generují části
JSON objektu v proceduře setParam
```

První tři proměnné využívá procedura reload(), ta do nich ukládá data, která jí jsou předána. Protože v ControlWebu nelze vytvořit vícerozměrné pole, které by pro toto bylo vhodnější, tak jsou všechny informace ukládány tak, že k sobě patří vždy ta data, která mají v těchto třech proměnných stejný index. Například ke klíči uloženém v poli names[0] patří hodnota v poli values[0] a čas v poli times[0].

Do proměnných html_text a param_text se již ukládají přímo části výsledného JSON objektu, které jsou generovány v procedurách show() a setParam(). Ostatní proměnné jsou v podstatě jen pomocné, protože bez nich by nebylo možné vygenerovat validní JSON objekt.

5.2 Procedury v ControlWebu zajišťující ukládání a přenos dat

V programu graphs je několik funkcí, které jsou vyvolávány v přesně daném pořadí, aby byla nakonec vygenerována odpověď pro webový prohlížeč, který jí dokáže zpracovat.

1) Při spuštění aplikace se zavolá procedura onStartup, ve které dojde k základní inicializaci proměnných.

2) Ještě z procedury onStartup je poprvé zavolána procedura _show z proměnnou prom_grafy.je_nacten = false. V každém dalším volání je již prom_grafy.je_nacten = true.

3) Při aktivaci přístroje program graphs v ControlWebu je zavolána procedura onActive, která jen spustí proceduru _reload()

4) V proceduře _reload je pro každou hodnotu, která se má uložit volána procedura safe(). Ta vždy uloží data do bufferu.

5) Při požadavku z prohlížeče na nová data je zavolána procedura render(), ve které se sestavuje JSON objekt odpovědi. Na začátku se nastaví pomocné proměnné a je zavolána procedura _show(). Po té se vymaže buffer a vygeneruje odpověď.

6) V proceduře _show() se nachází volání zobrazovacích procedur přístroje show() a procedury setParam(). Ty ukládají části odpovědí přímo do textových řetězců, které jsou nakonec kompletně sestaveny v proceduře render. Při prvním zavolání neudělá nic jiného, než jen spočte počet volání.

7) Body 3-6 se pak periodicky opakují, dokud nedojde k ukončení aplikace.

Důležité je vědět, že procedura reload() se volá v závislosti na periodě aktivace přístroje program. A nedělá nic složitého, jen uloží aktuální data do bufferu. Aktivaci přístroje je možné nastavit například 10x za sekundu.

Naopak procedura render() a následně procedura _show() slouží pro odeslání výstupu webovému prohlížeči a je volána periodicky dle požadavků z webového rozhraní. Standardně je tato perioda nastavena na frekvenci 4x za sekundu.

Procedura safe()

Procedura safe nedělá nic jiného, než že ukládá data do výše zmíněných proměnných. Výpočetně není nijak náročná, a proto může být volána s velkou frekvencí, aniž by to aplikaci nějak brzdilo. Celá funkce safe vypadá následovně:

```
procedure safe( name : string; value : real );
var
hodina, minuta, sekunda, msekunda : cardinal;
rok, mesic, den, den_v_tydnu : cardinal;
aktualni_cas : real;
begin
        // výpočet aktuálního času
date.GetTime(hodina, minuta, sekunda, msekunda);
date.GetDate(rok, mesic, den, den_v_tydnu);
aktualni_cas = (hodina*360+minuta*60+sekunda+msekunda*0.001);
        //uložení hodnot
```

```
        prom_grafy.names[prom_grafy.pocet_hodnot] = name;
        prom_grafy.values[prom_grafy.pocet_hodnot] = value;
        prom_grafy.times[prom_grafy.pocet_hodnot] = aktualni_cas;
        prom_grafy.pocet_hodnot = prom_grafy.pocet_hodnot + 1; // zvýšení
indexu
end_procedure;
```

Funkce je vždy volána z události onActive(). Frekvence vzorkování je závislá na časování celého programu.

Procedura setParam

Procedura setParam je nejjednodušší procedura v celém programu. Její jediný úkol je z dat, která jí jsou předána, vygenerovat část JSON objektu, který bude následně odeslán prohlížeči. Celá její definice vypadá takto:

```
procedure setParam( name, paramName, paramValue : string );
var
param_text : string;
begin
param_text = ',{';
param_text = param_text + 'name:"'+name+'",';
param_text = param_text + 'pn:"'+paramName+'",';
param_text = param_text + 'pv:"'+paramValue+'",';
param_text = param_text + '}';
prom_grafy.param_text = prom_grafy.param_text + param_text;
end_procedure;
```

Jednotlivý blok se nejprve ukládá do pomocné proměnné param_text a následně je tento blok přidán do hlavní proměnné prom_grafy.param_text, se kterou se dále pracuje v proceduře render.

Procedura show

Procedura show slouží pro vygenerování části JSON objketu, který nese údaje pro zobrazení přístroje. Přesněji tedy rozměry a umístění pozice objektu na stránce. Z předaných parametrů je rovnou vytvořena textová reprezentace objektu. Dále je také přečten buffer s uloženými hodnotami, které jsou také vložena do textové proměnné.

```
procedure show( name, type, owner : string; x, y, w, d : cardinal );
```

```
var
html_text : string;
value, time : real;
i : longcard;
carka : boolean;
begin

// jestliže již byla procedura poprvé zavolána, pak je možné ukládat
if prom_grafy.je_nacten then carka = false;

// vytváření textové reprezentace JSON objektu spojováním částí řetězců.
html_text = '{'; // začátek definice objektu
html_text = html_text + 'type :"' + type + '",';
html_text = html_text + 'owner :"' + owner + '",';
html_text = html_text + 'name :"' + name + '",';
html_text = html_text + 'x :' + str(x, 10) + ',';
html_text = html_text + 'y :' + str(y, 10) + ',';
html_text = html_text + 'w :' + str(w, 10) + ',';
html_text = html_text + 'd :' + str(d, 10) + ',';

html_text = html_text + 'val : ['; // začátek definice pole s hodnotami

// načtení uložených hodnot s bufferu
for i = 0 to prom_grafy.pocet_hodnot - 1 do

// najdeme klíč, který se shoduje s tím co je předán v argumentech funkce
if name = prom_grafy.names[i] then

        value = prom_grafy.values[i];  // načteme si odpovídající hodnotu
        time = prom_grafy.times[i]; // a čas

        // jestliže máme po elementu přidat čárku, pak jí přidáme
        if(carka) then html_text = html_text + ','; end;
        // přidáme záznam jako dvourozměrné pole s časem a hodnotou
        html_text = html_text + '[' + str(time, 10) + ',' + str(value,
10) + ']';
        carka = true; // čárka se nepřidává pouze při první iteraci
end;
end;
html_text = html_text + ']'; // ukončíme definici pole hodnot
html_text = html_text + '}'; // ukončíme definici objektu
// jestliže se nejedná o poslední zavolání funkce show, pak nakonec
přidáme čárku
```

```
if (prom_grafy.vystupni_index <> (prom_grafy.pocet_el-1)) then html_text
= html_text + ','; end;
```

```
prom_grafy.vystupni_index = prom_grafy.vystupni_index + 1; // a zvýšíme
počet volání o 1
// nakonec uložíme do hlavní proměnné
prom_grafy.html_text = prom_grafy.html_text + html_text;
else // v případě prvního volání vůbec jen spočteme počet volání
procedury show
prom_grafy.pocet_el = prom_grafy.pocet_el + 1;
end;
end_procedure;
```

Procedura render

K sestavení jednotlivých částí sloužily procedury show a setParam. Procedura render se stará o sestavení kompletního JSON objektu, který je touto procedurou vrácen a následně odeslán na výstup webové stránky. Zároveň také vymaže textové proměnné prom_grafy.html_text a prom_grafy.param_text, které využívaly předchozí procedury. Nakonec resetuje buffer pro ukládání hodnot, aby se mohla nová data ukládat od začátku a nezabíraly nové místo v paměti.

```
procedure render(): string;
var
html_text: string;
reload : cardinal;
begin
reload = 0; // proměnná určující zda má dojít k obnovení webové stránky
// inicializace proměnných použitých v procedurách show a setParam
prom_grafy.html_text = '';
prom_grafy.param_text = '';
prom_grafy.vystupni_index = 0;
// zavoláním funkce _show se naplní proměnné prom_grafy.html_text a
prom_grafy.param_text
_show();
if prom_grafy.pocet_hodnot > 1000000 then
  reload = 1; // v případě že dojde k přetečení bufferu obnov celou
stránku
end;
prom_grafy.pocet_hodnot = 0; // resetuje index pro ukládání dat do
bufferu
// vytvoříme HTML kód, který způsobí vytvoření JSON objektu a následné
zavolání funkce mojeGrafy.pridej ve webovém prohlížeči.
```

```
html_text = '<script type="text/JavaScript">';
html_text = html_text + 'data = {';
html_text = html_text + 'el : [';
html_text = html_text + prom_grafy.html_text; (* přečteme načtená data *)
html_text = html_text + prom_grafy.param_text; (* přečteme načtené
parametry *)
html_text = html_text + '],';
html_text = html_text + 'reload : ' + str(reload, 10);
html_text = html_text + '};';
html_text = html_text + 'if(window.mojeGrafy) ';
html_text = html_text + 'window.mojeGrafy.pridej(data);';
html_text = html_text + 'else ';
html_text = html_text + 'window.parent.mojeGrafy.pridej(data);';
html_text = html_text + '</script>';
return html_text;
end_procedure;
```

Vygenerovaný kód, který pak bude odeslán do webového prohlížeče, může v případě jednoho přístroje vypadat nějak takto:

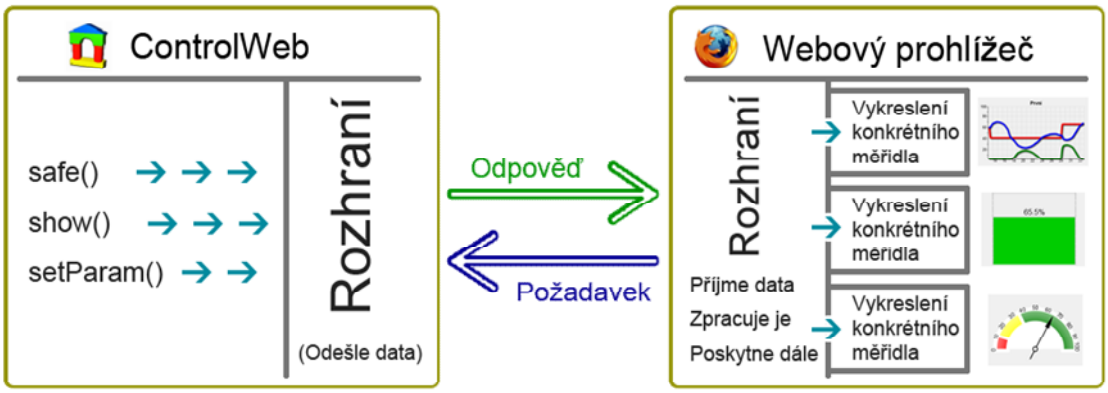
```
<script type="text/JavaScript">
data = {
el : [
        {
        type :"Meter", // v prohlížeči se vykreslí měřidlo typu meter
        owner :"panel_1", // nadřazený objekt je panel_1
        name :"meter_5", // jméno přístroje neboli klíč, který je uváděn
v proceduře show
        x :490, // pozice x vzhledem k nadřazenému elementu
        y :25, // pozice y vzhledem k nadřazenému elementu
        w :220, // šířka přístroje
        d :175, // výška přístroje
        val : [[10673.405,57.48],[10673.516,57.48]] // hodnoty načtené z
bufferu
                }
],
reload : 0 // Hodnota 1 nebo 0 dle toho, zda se má stránka obnovit
};
// ve webovém prohlížeči spustíme funkci mojeGrafy.pridej, které předáme
vygenerovaná data.
if(window.mojeGrafy) window.mojeGrafy.pridej(data);
else window.parent.mojeGrafy.pridej(data);
</script>
```

Reálný kód pak samozřejmě nebude obsahovat tabulátory ani odřádkování ale bude slepen na jednom řádku.

Na konci si všimněte, že se příkaz mojeGrafy.pridej volá jednou na stejném okně a jednou v nadřazeném. Tato podmínka je zde pro to, že při prvním volání je skript vložen do podřízeného iframu a proto je nutné přistoupit k nadřazenému oknu. V každém dalším volání je skript spuštěn již ve stejném okně jako celá knihovna, proto se nepřistupuje k nadřazenému oknu ale k aktuálnímu.

5.3 Propojení ControlWebu s webovým rozhraním

Pro tvůrce aplikací v ControlWebu je veškerá logika komunikace zastřešena pomocí připravených metod, které je potřeba volat, aby vše fungovalo tak jak má. Skutečná výměna informací však probíhá na základě JSON objektu [Úvod do JSON], který je vytvořen v ControlWebu a přenesen do prostředí webového prohlížeče. Tam je vyvolána JavaScriptová funkce pro obsluhu dat, která si přečte vše, co potřebuje a provede reálné vykreslení komponenty.



Obrázek 14: Schéma komunikace ControlWebu z webovým rozhraním a následně jednotlivými měřidly

Při pravidelném odeslání se do webového prohlížeče vždy odešle celý buffer, který je postupně plněn metodami safe(…) v proceduře _reload(). Pozice bufferu je v tomto okamžiku vynulována a je plněn od začátku. Každá hodnota je udržovaná v páru s časem, kdy byla zaznamenána. Tak je možné vykreslovat jak časové grafy, tak třeba jen ukazatele hodnot.

V prostředí webového prohlížeče taky existuje buffer, kde si archivuje vždy posledních 2000 zaznámů, které mohou být kdykoli vykresleny. Vzhledem k rozlišení obrazovek je počet 2000 bodů dostatečný.

Součástí odeslaných dat je také pozice a velikost přístroje spolu s nadřazeným objektem. Na webové stránce je pak vždy vyhledán nadřazený objekt, do kterého je vložen nový HTML5 přístroj a je umístěn na stejnou pozici jako ten původní, takže ho překryje a vznikne efekt, že byl přístroj nahrazen.

Naprogramovaná JavaScriptová knihovna mojeGrafy.js

JavaScriptová knihovna mojeGrafy.js zastřešuje veškeré dění ve webovém prohlížeči a slouží jako prostředník. Stará se o asynchronní načítání dat z ControlWebu a propojení s knihovou RGraph. V knihovně mojeGrafy jsou definovány všechny typy zobrazovaných přístrojů, které je možné použít. Definuje základní nastavení prvků z knihovny RGraph, tak, aby odpovídaly co nejvíce přístrojům v ControlWebu. Případně je může nějakým způsobem rozšířit o vlastní části.

Z hlediska tvůrce Aplikací v ControlWebu není nutné znát, co přesně se nachází v tomto souboru. Stačí jej připojit do hlavní stránky webu a o vše ostatní se již postará knihovna sama. Po prvním přijetí dat jsou inicializovány všechny objekty na stránce a umístěny na správnou pozici. Po té je pravidelně požádáno o načtení nových dat, která jsou použita pro překreslení všech objektů.

Součástí této knihovny je i oprava chyb nebo přesněji přemostění chyb, které obsahuje základní knihovna ControlWebu. Ta umožňuje interaktivní ovládání pouze v Internet Exploreru. Jiné prohlížeče nepodporuje. Po nasazení této knihovny bude webové rozhraní fungovat ve všech běžně používaných prohlížečích.

5.4 Rozhraní ve webovém prohlížeči

Jádro aplikace tvoří základní rozhraní mojeGrafy, které zapouzdřuje komunikaci mezi ControlWebem a jednotlivými přístroji, které se nakonec zobrazí na webové stránce. Rozhraní se stará o 4 základní věci.

1. Přijetí dat
2. Uložení dat do bufferu a jeho správa
3. Vytvoření elementů canvas pro kreslení se správnými rozměry a se správnou pozicí
4. Vyvolávání událostí přístrojů init a reload.

Díky tomu má programátor nových přístrojů usnadněnou práci při vytváření nových měřidel. Nemusí se totiž vůbec starat o to, jakým způsobem jsou data získána, ale pouze vykresluje měřidla do připravených objektů.

Přijetí dat

Předtím než je možné data přijmout, je nejprve nutné vytvořit žádost o zaslání nových data. Jedná se o klasicky GET požadavek webového serveru, který je realizován pomocí Ajaxu. Ihned po té, co je požadavek zpracován serverem v ControlWebu, je zpětně odeslán JSON objekt, který obsahuje všechny informace, které jsou pro vykreslení zapotřebí.

Objekt obsahuje jen 2 parametry. Prvním je pole objektů, které reprezentují buď zobrazení měřidla nebo parametry nastavené pomocí setParam v ControlWebu. Druhým parametrem objektu je boolean hodnota reload, která určuje, zda má dojít k tvrdému obnovení celé stránky.

Aplikace je navržena tak, že v poli objektů budou nejprve objekty zastupující zobrazené měřidla a teprve až za nimi předané parametry. Toho se pak také využívá při zpracovávání, kdy se nejprve vytvoří objekt, do kterého se ukládají veškerá přijatá data a až následně se nastavují přijaté parametry, které pak mohou využít jednotlivá měřidla.

Správa Bufferu

Jádro aplikace dále spravuje buffer, ze kterého čtou jednotlivá měřidla data. Ty jsou do něj ukládána postupně, podle toho, jak přijdou za sebou, a při dosažení 5000 hodnot je buffer vždy zkrácen na 2000 a ukládání pokračuje dál. V bufferu se tedy vždy nachází nejméně 2000 záznamů. Předpokladem ale je, že žádné měřidlo nevyužije všechna data ale vždy jen několik posledních. Už jen vzhledem k běžným rozlišením obrazovek by musel mít rozlišení nejméně 2000 px a měřidlo by na takovém monitoru muselo být maximalizované přes celou plochu.

Vytvoření objektů canvas

Objekt canvas je v podstatě plátno, na které je možné kreslit rastrovou grafiku. Jádro aplikace pro každé měřidlo vytvoří vlastní plátno dle rozměrů, které mu jsou zaslány a umístí je na správné pozice. Implementace jednotlivých měřidel, pak již jen kreslí do přidělených pláten a nehrozí žádná kolize jako například překreslování jednoho měřidla druhým.

Jednotlivé implementace měřidel ale nemusí být založené na rastrové grafice a objektu canvas. Nemusí jej tedy přímo využít. Z vytvořeného objektu je také možné vyčíst jeho rozměry a umístění, díky kterým může být měřidlo vykresleno jakkoli jinak pomocí jiných metod a objekt canvas bude jen sloužit pro zakrytí starého měřidla.

Volání událostí init a reload jednotlivých měřidel

Jádro aplikace, vždy když přijme nová data, prochází znovu všechny objekty uložené ve jmenném prostoru mojeGrafy.graphs a volá metody init popřípadě reload všech objektů, které zde nalezne. V případě, že objekt ještě není vykreslen, tak zavolá metodu init. Tato metoda je volána pouze jednou a implementace měřidla by zde měla provést úvodní implementaci.

V každé další iteraci se pak volá metoda reload, která má za úkol obnovit zobrazená data na měřidlu. O tom ale více v další kapitole.

6 Vytváření nových měřidel pro vykreslení

Aplikace je navržena tak aby bylo možné snadno rozšířit paletu základních měřidel o nová měřidla i bez širší znalosti základní aplikace. Jednotlivé měřila, která jsou vykreslována do webové stránky, jsou naprosto odstíněna od požadavků na nová data, jejich přijetí a zpracování. Dokonce jsou umístěna mimo hlavní kód v souboru meters.js.

Přidávání nových měřidel je tak mnohem snazší, jelikož se nemusí zasahovat do hlavního kódu a orientace v programu je tak mnohem jednodušší. Nově přidané měřidlo se pak stará pouze o vykreslení sebe sama (na plátno, které je mu dodáno) a všechna data, která k tomu potřebuje, mu jsou dodána.

6.1 Přidání nového měřidla

Všechna měřidla, se kterými rozhraní pracuje, jsou umístěna ve jmenném prostoru mojeGrafy.graphs. Momentálně existují 3 měřidla.

1. mojeGrafy.graphs.Line – pro vykreslování grafů
2. mojeGrafy.graphs.Meter – pro vykreslování ručičkového ukazatele
3. mojeGrafy.graphs.VProgress – pro vykreslení vertikální progresbaru

Jednotlivé názvy měřidel, pod kterými jsou uloženy do tohoto jmenného prostoru, jsou pak shodné jako s typem měřidla, který je vždy z ControlWebu předán metodou show. Samotné měřidlo je JavaScriptový objekt, který musí obsahovat 2 funkce, které jsou vyvolané základní aplikací.

1. init(obj) – Zavolá se jen jednou a slouží pro inicializaci přístroje
2. reload(obj) – Volá se periodicky dle frekvence obnovování a slouží pro opětovné vykreslení nových hodnot.

V argumentu funkce je předán objekt, ve kterém jsou uloženy základní informace, které je možné dále využít. Informace, které jsou v objektu uloženy jsou následující:

```
obj = {
name : name, // Klíč pod jakým je tento element volán z ControlWebu (bez
indexu)
canvasId : id, // Id objektu canvas
canvas : c, // Přímo element canvas
type : type, // Typ objektu, který se má vykreslit
chart : null, // Prostor pro uložení objektu, který reprezentuje
vykreslené měřidlo
val : value, // Vždy dvourozměrné pole hodnot, které se mají vykreslit
params : {}, // Objekt s parametry, které jsou nastavené v ControlWebu
pomocí setParam
isInited : false // Boolean hodnota určující zda už bylo zavoláno init
}
```

V tomto objektu je předáno jak id elementu, tak přímo objekt canvas. Je to hlavně z důvodu usnadnění práce, protože je někdy výhodnější pracovat přímo s objektem a někdy

je výhodnější znát jeho id. S pomocí jednoho je sice snadné získat druhé, ale hlavní aplikace to udělá za nás.

Parametr type, obsahuje řetězec označující název přístroje, který se má vykreslit. V případě grafu bude obsahovat „Line“. Stejný řetězec je pak druhým parametrem u funkce show v ControlWebu.

Parametr chart je pak pouze vytvořený prostor, do kterého si můžete uložit nějaký svůj objekt, který bude reprezentovat vykreslený objekt. Základní měřidla využívají knihovnu RGraph a do tohoto parametru jsou pak ukládány právě instance vytvořených objektů z knihovny RGraph.

Nejdůležitější částí je pak parametr val, ve kterém je vždy uloženo nejméně 2000 naposledy uložených hodnot. Jestliže jsou hodnoty pouze jednorozměrné, pak jsou uloženy pod indexem 0. Jestliže jsou v ControlWebu ukládány s indexy, pak se k nim přistupuje vždy pod indexy, s jakými byly v ControlWebu uloženy.

V objektu params jsou uloženy všechny parametery, jaké byly nastaveny v controWebu metodou setParam.

Ukázka objektu, který bude předán funkcím init a reload

Objekt obj obsahuje vše, co je mu předáno z ControlWebu. Pro snazší pochopení je zde znázorněn příklad jednoho měřidla s následujícím nastavením.

Funkce _reload()

```
safe('data', round(meter_5.GetValue()*100)/100);
```

Funkce _show()

```
meter_5.GetRect( true, x, y, w, d);
show('data', 'Meter', meter_5.GetOwner(), x, y, w, d);
setParam('data', 'minValue', 0);
setParam('data', 'maxValue', 100);
```

Objekt obj dle předchozího nastavení

```
obj = {
```

```
name : 'data', // První parametr funkce show
canvasId : id, // Vygenerované id (vytváří se na straně prohlížeče)
canvas : c, // Reprezentace objektu cavas (vytváří se na straně
prohlížeče)
type : 'Meter', // Druhý parametr funkce show
chart : null, // Prostor pro uložení objektu
val : [[…, …, …., ….]], // jednotlivé hodnoty uložené s pomocí funkce
safe
params : {
minValue : 0,
maxValue : 100
}, // Parametry nastavené pomocí setParam
isInited : false // Při zavolání init je false. Při volání reload je už
true
}
```

Přístup k hodnotám

Všechny hodnoty jsou v parametru val uložené v párech s časem, kdy byly zaznamenány. Chcete-li například zjistit poslední uloženou hodnotu, pak je k ní možné přistoupit tímto kódem:

```
var v = el.val[0][el.val[0].length-1][1]; // index 1 na konci obsahuje
hodnotu
```

Jedná se zde o hodnoty s indexem 0.

Podobně lze zjisti čas, kdy byl poslední záznam pořízen.

```
var v = el.val[0][el.val[0].length-1][0]; // index 0 na konci obsahuje
čas pořízení
```

Čas je vždy uložen v sekundách od začátku dne.

Kam vložit kód nového měřidla?

Nové měřidlo můžete přidat buď přímo do souboru meters.js, kde jsou všechna základní měřidla nebo jej můžete přidat do vlastního souboru, který pak jen načtete. Pro druhý případ existuje v základu funkce addScript, která má následující syntaxi:

```
addScript(adresa_souboru, callback_funkce_volána_po_načtení)
```

Tuto funkci je také možné použít v případě, kdy je nějaký kód závislý na nějaké další externí knihovně. Všechny základní měřidla využívají knihovnu pro vykreslování grafu RGraph. Proto je také přidání všech měřidel obsluhováno touto funkcí. Například měřidlo pro vykreslování grafů je přidáno následovně:

```
mojeGrafy.addScript('mojeGrafy/libraries/RGraph.line.js', function(){
… Obsah funkce, která se zavolá po načtení souboru
'mojeGrafy/libraries/RGraph.line.js' …
});
```

Mějme například soubor s novým měřidlem pojmenovaný meridlo.js a umístěné ve stejném adresáři jako soubor meters.js, pak pro jeho načtení přidejte na konec souboru meters.js tento řádek:

```
mojeGrafy.addScript('mojeGrafy/meridlo.js');
```

Callback funkce není povinná a pro tento případ je i zbytečná, proto ji ani neuvádím. Vše v souboru meridlo.js bude automaticky provedeno hned po načtení. Funkci addScript můžete předat také více adres současně, tak že jejich adresy oddělíte čárkou. Volání pak může vypadat takto:

```
mojeGrafy.addScript('mojeGrafy/libraries/RGraph.vprogress.js,
mojeGrafy/libraries/RGraph.meter.js', function(){ … });
```

Všimněte si, že adresy nejsou předávány jako dva řetězce, ale pouze jako jeden, ve kterém jsou dvě adresy. Tato funkce také ošetřuje případy, kdy je již soubor jednou nahrán a bude požadován podruhé. V takovém případě se již znovu nestahuje, ale je rovnou zavolána callback funkce.

6.2 Ukázka implementace LED displeje

Pro názornost je mnohem snazší předvést implementaci v praxi. Vytvoříme si proto jednoduché měřidlo jako led displej, který je jako standardní měřidlo v knihovně rGraph a celý kód naprogramujeme v novém souboru.

Hned na začátku si vytvoříme vedle souboru meters.js nový prázdný soubor led.js. Ihned poté si otevřeme soubor meters.js a na konec callback funkce pro načtení jádra knihovny rGraph vložíme tento řádek:

```
mojeGrafy.addScript('mojeGrafy/led.js');
```

Tím jsme zařídili, že soubor led.js bude načten a zpracován.

Dále si otevřeme opět soubor led.js a budeme pokračovat v implementaci nového měřidla. Protože je implementace závislá na externích knihovnách, tak využijeme funkci addScript pro jejich vložení do stránky.

```
mojeGrafy.addScript('mojeGrafy/libraries/RGraph.led.js', function(){
…
});
```

Prvním parametrem je adresa k JavaScriptovému souboru, na kterém je další kód závislý. Druhý parametr je funkce, která bude zavolána v okamžiku, kdy bude soubor načtený a zpracovaný. Při využívání knihovny rGraph je vždy nutné zahrnou základní soubor jádra a dále soubor, který se stará o zobrazení konkrétního grafu. Načtení samotného jádra je však již řešeno v souboru meters.js a my se tím nemusíme zabývat.

Celá implementace, pak vypadá následovně:

```
mojeGrafy.addScript('mojeGrafy/libraries/RGraph.led.js', function(){
        mojeGrafy.graphs.Led = { // do jmeného prostoru graphs si vložíme
nový objekt Led
                init : function(el){ // funkce pro inicializaci objektu
                        // načteme si poslední hodnotu z bufferu
                        var v = el.val[0][el.val[0].length-1][1];
                        // vytvoříme novou instanci RGraph.LED objektu
                        var led = new RGraph.LED(el.canvasId,
v.toString());
                        // nastavíme potřebné parametry
                        led.Set('chart.zoom.hdir', 'center');
                        led.Set('chart.zoom.vdir', 'center');
                        led.Set('chart.light', 'red');
                        // nastavíme také všechny parametry předané pomocí
setParam
                        for(var k in el.params){
                                var v = mojeGrafy.getValue(el.params[k]);
                                led.Set(k, v);
                        }
                        led.Draw(); // a vykreslíme LED displej
```

```
                        el.chart = led; // nakonec si uložíme instanci do
proměnné chart
                },
                reload : function(el){ // funkce volána při obnovení dat
                   var v = el.val[0][el.val[0].length-1][1];  // opět si
načteme poslední hodnotu
                   el.chart.text = v.toString(); // a nastavíme ji
měřidlu
                   el.chart.Draw(); // nakonec jej opět překreslíme
                }
        };
});
```

Všimněte si, že objekt, který je uložíte do el.chart ve funkci init je pak dostupný opět jako el.chart ve funkci reload. Tento parametr existuje právě pro snadné předávání této instance mezi těmito dvěma funkcemi.

6.3 Funkce, které se mohou hodit při vykreslování

V základu jsou k dispozici 4 funkce, které se mohou hodit při práci s tímto rozhraním. Tyto funkce nejsou nikterak složité, ale zefektivňují práci s jednotlivými grafy.

mojeGrafy.getValue(void value) – Funkci je možné předat řetězec nebo číslo. Jestliže se jí předá číslo, pak neudělá nic jiného, než že vrátí to samé číslo. V případě, že se jí předá řetězec, který reprezentuje číslo, pak vrátí číslo. Jestliže se jí předá řetězec, který nereprezentuje číslo, pak vrátí ten samý řetězec. Usnadňuje načítání čísel z přijatého objektu, kdy jsou některé parametry předány jako řetězec avšak ve skutečnosti to jsou číselné hodnoty.

mojeGrafy.clearCanvas(string id) – Tato funkce vymaže nebo přesněji řečeno překreslí objekt canvas jehož id jí bude předáno.

mojeGrafy.fillCanvas(string id, string color) – Funkce fillCanvas překreslí objekt canvas jehož id jí bude předáno zadanou barvou v druhém argumentu.

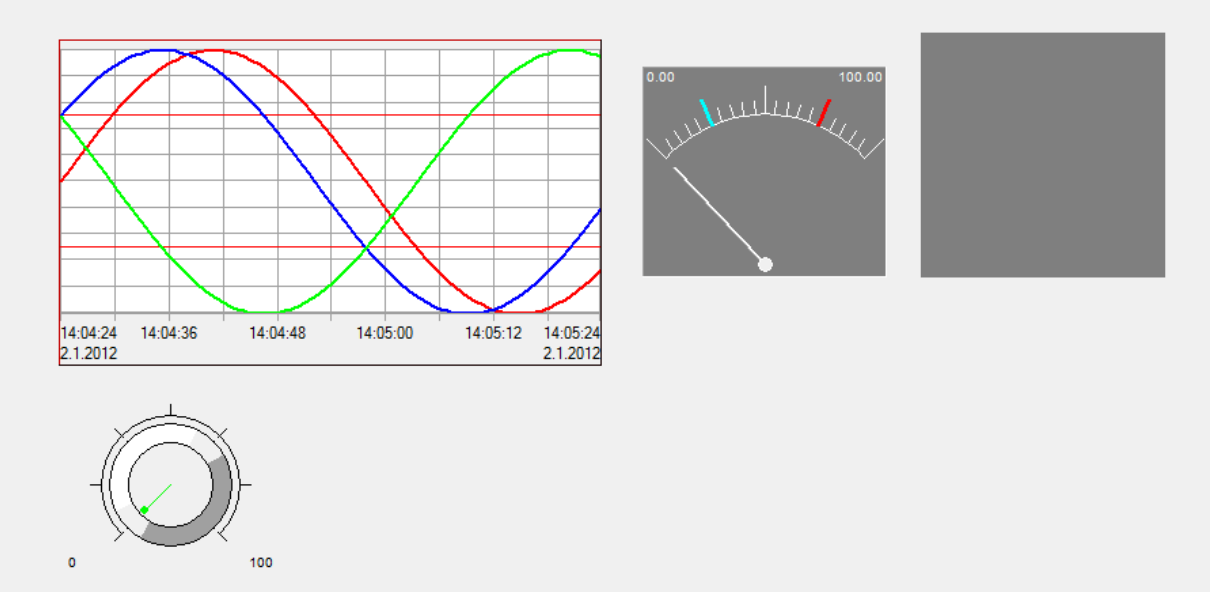
mojeGrafy.addScript(string url, function callback) – o této funkci jsem se zmiňoval již výše. Tato funkce načte všechny skripty předané v prvním parametru a po úspěšném načtení zavolá callback funkci.

7 Demonstrační úloha

Pro demonstraci funkčnosti tohoto řešení jsem navrhl jednoduchou úlohu, která obsahuje jeden řídící prvek a 3 různé způsoby zobrazení.

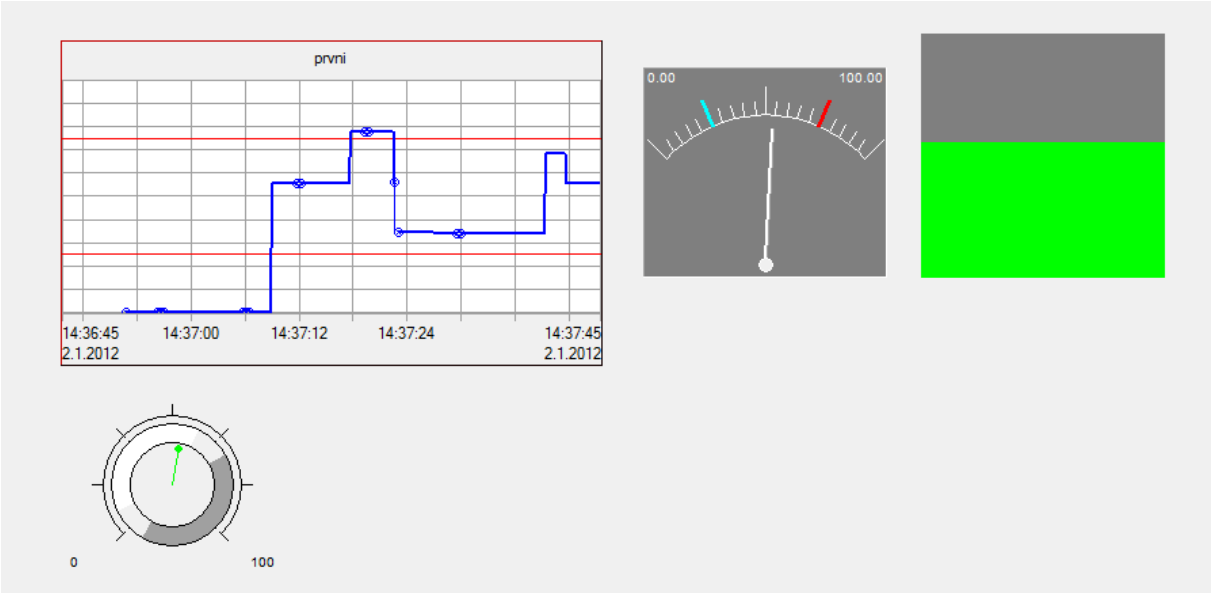
1. Graf
2. Ručičkové měřidlo
3. Vertikální progresBar

Navržená demonstrační úloha je na obrázku 15.



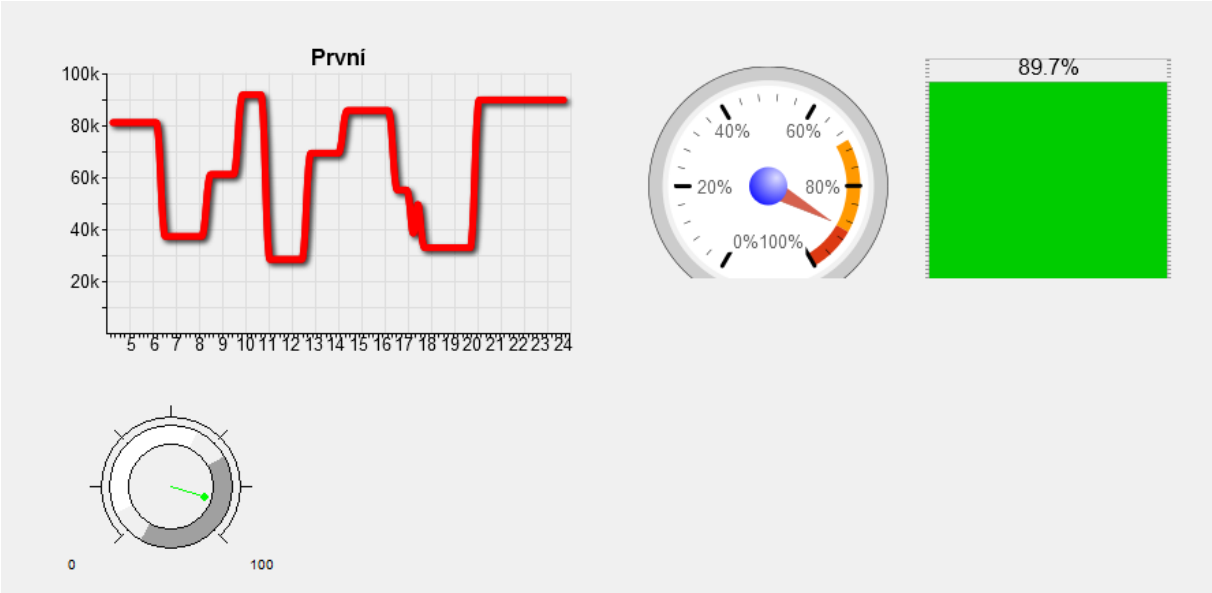
Obr. 15: Demonstrační úloha v prostředí ControlWeb

Pro tuto úlohu jsem si pomocí průvodce vytvořil webové rozhraní, které je zobrazeno na obrázku 4 a které je vzhledově naprosto totožné s rozhraním v počítači. Nastavil jsem pravidelné obnovování webu, aby bylo možné sledovat změnu hodnot při nastavení potenciometru.



Obr. 16:Demonstrační úloha zobrazená ve webovém prohlížeči

Na tuto úlohu jsem aplikoval změnu vizualizace pomocí HTML5 a knihovny RGraph tak, že webová stránka se nemusí pravidelně obnovovat. Všechny ukazatele jsou překreslovány 4x za vteřinu a díky knihovně RGraph jsou lépe designově vykresleny než původní rozhraní. Výsledek si můžete prohlédnout na obrázku 17.



Obr. 17: Demonstrační úloha vykreslena pomocí HTML5 a knihovny RGraph

Z hlediska tvůrce tohoto rozhraní spočívá změna pouze v dopsání několika řádků do programu v ControlWebu a přilinkování JavaScriptové knihovny do vygenerované HTML

stránky. Znalost tvorby webových stránek ani HTML kódu není kromě vepsání jednoho řádku zapotřebí.

7.1 Implementace HTML5 rozhraní do ControlWebu

Cílem této práce je, aby implementace nového webového rozhraní byla pro tvůrce aplikací co nejsnazší. Celá změna spočívá v pěti krocích. Nejprve je však nutné vytvořit si webové rozhraní pomocí průvodce v ControlWebu. Po té již je možné aplikovat tyto kroky:

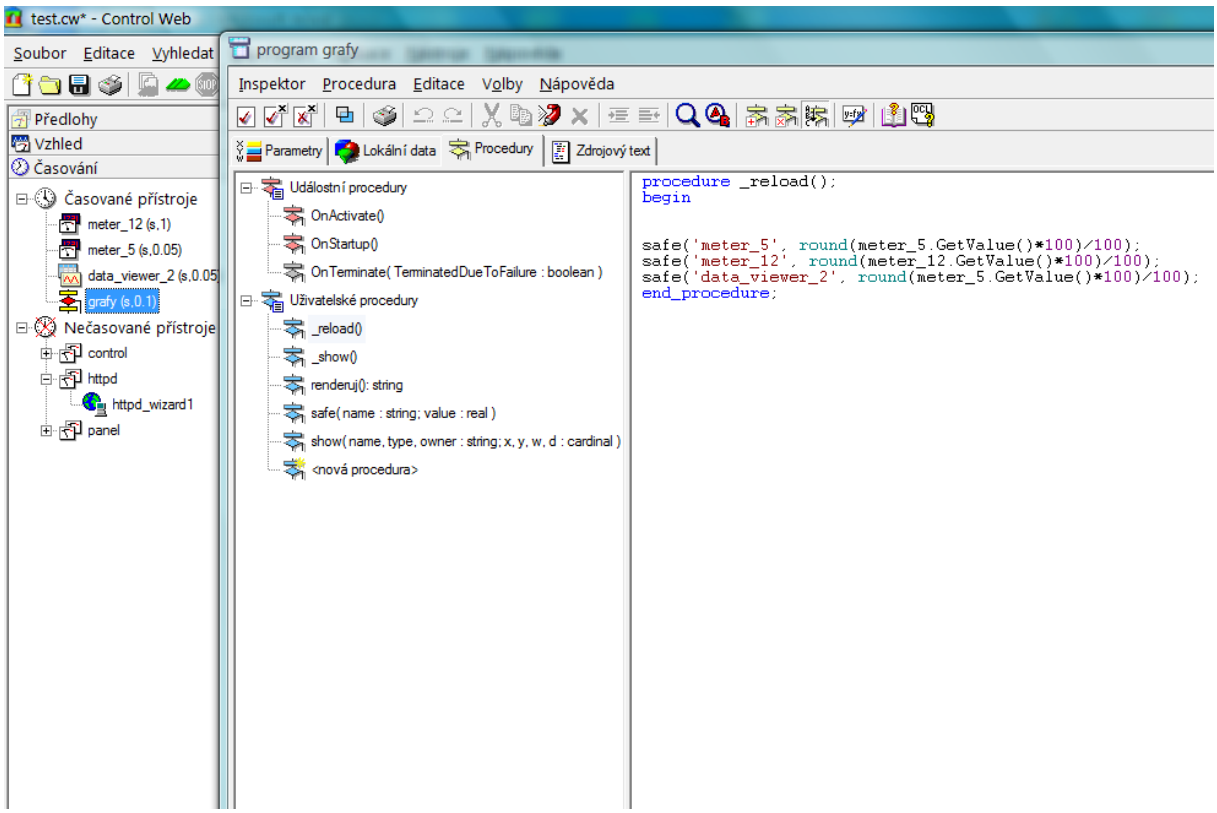
1. Do adresáře, kde byly vytvořeny webové soubory nakopírovat složku mojeGrafy
2. Do souboru main.htm v části <head>…(sem)</head>vložit následující řádek:

```
<script type="text/JavaScript" src="mojeGrafy/mojeGrafy.js"></script>
```

3. Do kódu v ControlWebu nakopírovat kód v příloze 1.
4. Otevřít si přístroj httpd a nakonec procedury GenerateStatusFrame dopsat řádek:

```
PutHTML(graphs.render());
```

Tím máme hotovou přípravu. Pátým a posledním krokem je stanovit co přesně má být nahrazeno a odkud budou brána data pro zobrazení. Po nakopírování kódu z přílohy 1 vznikl v seznamu přístrojů nový program grafy. Ten si otevřeme a zobrazíme si procedury.



Obr. 18: Program grafy a procedura _reload()

Nás budou zajímat procedury _reload() a _show(). První jmenovaná se volá pravidelně 10x za sekundu. Toto volání je možné změnit nastavením jiné periody časování programu. V této proceduře probíhá ukládání změřených dat do bufferu. Vždy je nutné uvést klíč podle, kterého budou data uložena a samotnou hodnotu. Jako klíč je vhodné pro přehlednost zvolit přímo název měřeného přístroje, ale není to nutné.

Ukládání se provádí pomocí metody safe s následující syntaxí:

```
safe(klíč : string; hodnota : real);
```

V proceduře _reload() je nutné uvést všechny proměnné, které mají být zaznamenány pro zobrazení na webu.

Druhá procedura s názvem **_show()** je volána vždy, když se aplikace na webu dožaduje informací z ControlWebu v našem demonstračním příkladě je to 4x za sekundu. Zobrazování informací pro web obstarává metoda show, která potřebuje znát klíč k hodnotám, které se mají zobrazit, typ přístroje, který se má zobrazit, nadřazený přístroj (nejčastěji panel) a rozměry s pozicí x, y samotného přístroje.

Metoda show má následující syntaxi:

```
show(klíč, typ_přístroje, název_nadřazeného_přístroje : string; x, y,
šířka, výška: cardinal);
```

Příklad použití může být následující:

```
meter_5.GetRect( true, x, y, w, d);
show('meter_5', 'Gauge', meter_5.GetOwner(), x, y, w, d);
```

Stejně jako v předchozím případě, tak i v proceduře _show() musí být uvedeny všechny přístroje, které mají být na webu zobrazeny.

Po splnění všech těchto kroků již bude vykreslení na webu zcela automatické. Přístroje se vykreslí na správných místech a se správnými rozměry. Způsob jakým se dostanou data z ControlWebu do webového rozhraní se nemusí vůbec řešit.

Nastavování parametrů pomocí funkce setParam

Jednotlivá měřidla je možné různě přizpůsobovat pomocí různých parametrů. Pro různé typy měřidel se mohou nastavitelné parametry lišit a ty, které je možné nastavit, vždy záleží na konkrétní implementaci měřidla ve webovém prohlížeči. Jejich seznam je pak uveden v dokumentaci.

Parametry měřidel lze nastavit v ControlWebu pomocí funkce setParam, která má následující syntaxi:

```
setParam(klíč, název_parametru, hodnota_parametru)
```

Konkrétní použití pro nastavení minima a maxima u měřidla typu Gauge je pak následující:

```
setParam('meter_5', 'minValue', '0'); // nastavení minima
setParam('meter_5', 'maxValue', '100'); // nastavení maxima
```

Takto lze pak nastavit neomezené množství parametrů různým typům měřidel a každé měřidlo lze pak snadno rozšířit o další možné parametry.

7.2 Program demonstrační úlohy

V demonstrační úloze jsou použity celkem 3 měřidla, proto je kód předchozích metod o něco větší.

Procedura _reload()

```
procedure _reload();
begin

safe('meter_5', round(meter_5.GetValue()*100)/100);
safe('meter_12', round(meter_12.GetValue()*100)/100);
safe('data_viewer_2', round(meter_5.GetValue()*100)/100);

end_procedure;
```

Procedura _show()

```
procedure _show();
var
x, y, w, d : longint;
begin
meter_5.GetRect( true, x, y, w, d);
show('meter_5', 'Gauge', meter_5.GetOwner(), x, y, w, d);
meter_12.GetRect( true, x, y, w, d);
show('meter_12', 'VProgress', meter_12.GetOwner(), x, y, w, d);
data_viewer_2.GetRect( true, x, y, w, d);
show('data_viewer_2', 'Line', data_viewer_2.GetOwner(), x, y, w, d);
end_procedure;
```

Zapsání tohoto kódu a splnění všech předchozích kroků způsobí nahrazení daných přístrojů jejich zadanými alternativami z knihovny RGraph.

Závěr

Vizualizace procesů ve webovém prohlížeči je v ControlWebu standardně řešena pomocí obrázku, který je jím vygenerován a následně odeslán do webového prohlížeče, který jej pouze vykreslí a nijak jinak s ním nepracuje. Takové řešení je poměrně snadné na vytvoření ale má několik nevýhod. Přenesení obrázku po síti je datově náročné, generování obrázku může být při běhu realtime aplikace náročné i časově, webová stránka se musí pravidelně obnovovat a narušuje tak uživatelský komfort, frekvence obnovování nemůže být příliš velká (přibližně méně než sekundu) a je toho více. Z toho důvodu jsem vytvořil webové rozhraní, které tyto problémy odstraňuje.

Předtím než jsem začal toto řešení vytvářet, tak jsem si důkladně prostudoval veškeré možnosti technologie HTML5 a většinu z nich jsem zde i popsal. Součástí mého studia bylo i vytvoření několika demonstračních úloh, na kterých je vidět jakým způsobem se posunul pohled na webový prohlížeč. Ten již není pouze programem, který má nějakým způsobem poskládat a zobrazit přijatý HTML kód ale stává se prostředím, ve kterém můžou běžet aplikace velmi podobné těm, které známě z běžného operačního systému. V další části se pak využívá hlavně element canvas dohromady s knihovnou pro generování grafů RGraph.

Hlavní práce spočívala v nahrazení současného statického rozhraní, které je potřeba pravidelně obnovovat, rozhraním dynamickým, které využívá možností HTML5. Hlavním rozdílem oproti původní variantě je, že se na web nepřenáší obrázky, které jsou datově poměrně velké ale pouze data pro vykreslení a samotné vykreslování probíhá až na straně webového prohlížeče.

Nejprve jsem se věnoval implementaci nového rozhraní v ControlWebu, kde je hlavně řešeno, jakým způsobem se má poskládat odpověď na požadavek o nová data z webového prohlížeče. Následně je rozpracována část, kdy jsou data přijatá do prohlížeče, který je nějak zpracuje a vykreslí.

Při své práci jsem pro vykreslování grafů a ukazatelů využil volně dostupné knihovny RGRaph, která využívá nového HTML5 elementu canvas pro vykreslování rastrové grafiky. Současně také zastřešuje rozdíly v implementaci mezi prohlížeči a její součástí je i emulace canvasu v Internet Exploreru verze menší než 9, který jej nepodporuje.

Klíčovou roli hraje mnou naprogramovaná knihovna mojeGrafy.js, která zprostředkovává komunikaci mezi ControlWebem a vykresluje jednotlivé přístroje pomocí knihovny RGraph. Nastavuje výchozí podobu elementu a může je také libovolně doplnit, aby co nejvíce odpovídali přístrojům v ControlWebu. V současné době jsou naprogramované 3 přístroje, kterými jsou vertikální progressBar, ručičkový ukazatel a graf. Tyto 3 přístroje jsou také předvedeny v demonstrační úloze.

Nové rozhraní má několik výhod.

1. Menší datový tok od serveru k uživateli => menší nároky na síť.
2. Díky menšímu datovému toku je možné data odesílat častěji.
3. Webová stránka se neobnovuje celá, ale jsou pouze překreslovány jednotlivé části.
4. Z uživatelského hlediska se webová aplikace (z hlediska reakcí) jeví téměř jako desktopová aplikace.

Z hlediska tvůrce uživatelských rozhraní v ControlWebu je celé rozšíření naprogramováno tak, aby z jeho implementací bylo co nejméně práce. Kdy běžný uživatel nemusí vůbec vědět, jakým způsobem rozhraní pracuje, ale pouze jakým způsobem jej má použít.

Ke konci práce se pak věnuji možnosti rozšíření aplikace o nová měřidla. Díky navrženému rozhraní je přidání nových měřidel velice snadné. Každé měřidlo je jakýsi modul, který se stará o vykreslení sebe sama a hlavní jádro tomuto modulu pouze poskytne potřebná data a vytvoří prostředí, ve kterém může pracovat.

Závěrečná demonstrativní úloha ukazuje, jakým způsobem lze nahradit běžně generované statické rozhraní ControlWebu novým a dynamickým rozhraním využívající technologie HTML5.

Seznam použité literatury

AJAX. *Wikipedia* [online]. [cit. 2012-01-04]. Dostupné z:

http://cs.wikipedia.org/wiki/AJAX

Cascading Style Sheets (CSS) Snapshot 2010. *W3C* [online]. [cit. 2011-06-24]. Dostupné z: http://www.w3.org/TR/CSS/

CSS Fonts Module Level 3. *W3C* [online]. 2011 [cit. 2011-06-24]. Dostupné z WWW: <http://www.w3.org/TR/css3-fonts/>.

CSS3 Transform. *ZEN Elements.* [online]. 2010 [cit. 2011-06-24]. Dostupné z: http://www.zenelements.com/blog/css3-transform/.

Css3PIE [online]. [cit. 2011-06-25]. Dostupné z: http://css3pie.com/

Drag Operations. *Mozilla Developer Network* [online]. 8.4.2010 [cit. 2011-06-25].

Dostupné z: https://developer.mozilla.org/En/DragDrop/Drag_Operations

Dudek Jan. CSS3 - kaskádové styly budoucnosti. *Interval.cz* [online]. 18. 05. 2005 [cit. 2011-06-24]. Dostupné z: http://zdrojak.root.cz/clanky/webdesigneruv-pruvodce-po-html5-tahame-data-od-navstevnika/.

Dvořák Jakub. HTML 5: nová generace webů. *Živě.cz* [online]. 9. 7. 2009 [cit. 2011-06-24]. Dostupné z: http://www.zive.cz/clanky/html-5-nova-generace-webu/sc-3-a-147815/default.aspx

Geolokace. *Wikipedia* [online]. 7.4.2011 [cit. 2011-06-25]. Dostupné z: http://cs.wikipedia.org/wiki/Geolokace.

Hassman Martin. Začínáme s HTML5 canvasem. *Zdroják.cz* [online]. 10. 4. 2009 [cit. 2011-06-24]. Dostupné z: http://zdrojak.root.cz/clanky/zaciname-z-html5-canvasem/.

HTML 5 differences from HTML 4. *W3C* [online]. 2009 [cit. 2011-06-24]. Dostupné z:

http://www.w3.org/TR/2009/WD-html5-diff-20090423/.

HTML Living Standard. *Web Hypertext Application Technology Working Group* [online]. 24.6.2011 [cit. 2011-06-25]. Dostupné z WWW: <http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#microdata>.

HTML5 video. *Wikipedie* [online]. 16.4.2011 [cit. 2011-06-24]. Dostupné z:

http://cs.wikipedia.org/wiki/HTML5_video.

Malý Martin. AppCache: webové aplikace i bez připojení. *Zdroják.cz* [online]. 27.7.2010 [cit. 2011-06-24]. Dostupné z: http://zdrojak.root.cz/clanky/appcache-webove-aplikace-i-bez-pripojeni/.

Malý Martin. Geolokace v prohlížeči. *Zdroják.cz* [online]. 29.4.2010 [cit. 2011-06-25]. Dostupné z: http://zdrojak.root.cz/clanky/geolokace-v-prohlizeci/.

Malý Martin. HTML5 Audio: rádio ve vašich stránkách. *Zdroják.cz* [online]. 13.7.2010 [cit. 2011-06-24]. Dostupné z: http://zdrojak.root.cz/clanky/html5-audio-radio-ve-vasich-strankach/.

Malý Martin. HTML5: ukládáme si data k elementům. *Zdroják.cz* [online]. 6.12.2010 [cit. 2011-06-24]. Dostupné z: http://zdrojak.root.cz/clanky/html5-ukladame-si-data-k-elementum/

Malý Martin. Webdesignérův průvodce po HTML5: Databáze v prohlížečích. *Zdroják.cz* [online]. 17.8.2010 [cit. 2011-06-24]. Dostupné z: http://zdrojak.root.cz/clanky/webdesigneruv-pruvodce-po-html5-databaze-v-prohlizecich/.

Malý Martin. Webdesignérův průvodce po HTML5: Multithreading s WebWorkers. *Zdroják.cz* [online]. 10.8.2010 [cit. 2011-06-25]. Dostupné z: http://zdrojak.root.cz/clanky/webdesigneruv-pruvodce-po-html5-multithreading-s-webworkers/.

Malý Martin. Webdesignérův průvodce po HTML5: Táhni a srůstej. *Zdroják.cz* [online]. 5.1.2011 [cit. 2011-06-25]. Dostupné z: http://zdrojak.root.cz/clanky/webdesigneruv-pruvodce-po-html5-tahni-a-srustej/

Malý Martin. Webdesignérův průvodce po HTML5: WebStorage. *Zdroják.cz* [online]. [cit. 2011-06-24]. Dostupné z: http://zdrojak.root.cz/clanky/webdesigneruv-pruvodce-po-html5-webstorage/.

Malý Martin. Webdesignérův průvodce po HTML5: WebStorage. *Zdroják.cz* [online]. 3.8.2010 [cit. 2012-05-17]. Dostupné z: http://zdrojak.root.cz/clanky/webdesigneruv-pruvodce-po-html5-webstorage/

Milušek Martin. HTML5: První krůčky s FileSystem API. *Zdroják.cz* [online]. 15.2.2011 [cit. 2011-06-24]. Dostupné z: http://zdrojak.root.cz/clanky/html5-prvni-krucky-s-filesystem-api/.

Sládek Jan. Webdesignérův průvodce po HTML5 - pohyblivé obrázky. *Zdroják.cz* [online]. 15.6.2010 [cit. 2011-06-24]. Dostupné z: http://zdrojak.root.cz/clanky/webdesigneruv-pruvodce-po-html5-pohyblive-obrazky/.

Sládek Jan. Webdesignérův průvodce po HTML5 - používáme pohyblivé obrázky. *Zdroják.cz* [online]. 22.6.2010 [cit. 2011-06-24]. Dostupné z: http://zdrojak.root.cz/clanky/webdesigneruv-pruvodce-po-html5-pouzivame-pohyblive-obrazky/.

Sládek Jan. Webdesignérův průvodce po HTML5 - taháme data od návštěvníka. *Zdroják.cz* [online]. 29.6.2010 [cit. 2011-06-24]. Dostupné z: http://zdrojak.root.cz/clanky/webdesigneruv-pruvodce-po-html5-tahame-data-od-navstevnika/

The RGraph documentation and examples [online]. [cit. 2012-01-04]. Dostupné z: http://www.rgraph.net/docs/index.html

Úvod do JSON. *JSON* [online]. [cit. 2012-01-04]. Dostupné z WWW: http://json.org/json-cz.html

Nápověda ControlWebu